

COMMODORE 64=

PROGRAMMEREN IN MACHINETAAL



M.B. IMMERZEEL

DE MUIDERKRING

C
B
M



C
B
M



C
B
M

COMMODORE 64

PROGRAMMEREN IN MACHINETAAL

© 1985 De Muiderkring b.v. Bussum - Nederland

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotocopie, microfilm of op welke andere wijze ook, zonder voorafgaande toestemming van de uitgever.

ISBN 90 6082 256 0

M. B. IMMERZEEL

COMMODORE 64

**PROGRAMMEREN
IN MACHINETAAL**



DE MUIDERKRING B.V. - BUSSUM

UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

INHOUD

DEEL 1. DE COMPUTER, ALGEMEEN

1. De opbouw van de computer	8
1.1. De samenstellende delen	8
1.2. Het hexadecimale codsysteem	9
1.3. Indeling van het RAM geheugen.....	10
1.4. Sprongopdrachten	11
1.5. Basisprogramma's voor logische functies.....	12
1.6. De 6510 CPU	13
2. Instructies	18
2.1. De instructieset	18
2.2. Transportinstructies	18
2.3. De rekeninstructies.....	22
2.4. De vergelijkingsinstructies.....	25
2.5. Logische instructies	26
2.6. De schuifinstructies	27
2.7. Voorwaardelijke spronginstructies (Branch)	28
2.8. Onvoorwaardelijke spronginstructies (Jump)	31
2.9. Overige	37
3. Niet geïndexeerde adresseermethoden	38
3.1. Immediate addressing.....	38
3.2. Zero page addressing	38
3.3. Absolute addressing	39
3.4. Relatieve addressing	39
3.5. De accu en Implied addressing	40
3.6. De accu en Implied addressing	41
4. Geïndexeerde adresseermethoden.....	42
4.1. Zero page index addressing.....	42
4.2. Absolute index addressing.....	43
4.3. De (indirect,x) addressing.....	43
4.4. De (indirect,y) addressing.....	44

DEEL 2. DE COMMODORE 64

5. Inleiding	46
5.1. Het invoeren van machinetaalprogramma's.....	46
5.2. Het save en loaden van machinetaalprogramma's	48
5.3. Het gebruiken van de geheugenruimte	49
6. Het in- en uitvoeren van variabelen	56
6.1. Inleiding	56
6.2. Het in- en uitvoeren van karakters	56
6.3. Het werken met het toetsenbord.....	58

6.4. Het printen van stringvariabelen	61
6.5. Het invoeren van floating point variabelen	62
6.6. Het laden van geheugenplaatsen en het printen van hun inhoud.....	64
7. Saven, loaden en printen	67
7.1. Het saven en loaden van programma's	67
7.2. Het saven en loaden van een file naar de disk	68
7.3. Het saven en loaden van een file naar de tape	70
7.4. Karakters naar de printer	72
7.5. Hard copy van het scherm.....	73
8. Bewerkingen met getallen	76
8.1. Rekenkundige en logische bewerkingen.....	76
8.2. Functies	78
8.3. Het toevalsgetal en pi.....	79
8.4. Voorbeelden van rekenkundige bewerkingen.....	80
8.5. De USER functie.....	82
8.6. Het vergelijken van twee getallen.....	82
9. Grafische mogelijkheden.....	84
9.1. Het printen.....	84
9.2. Bewegende beelden.....	85
9.3. Het gebruik van de sprites.....	87
9.4. Scrolling.....	89
9.5. Grafische functies met hoogoplossend vermogen.....	94
Lijst van gebruikte subroutines.....	105
Overzicht lijsten	106
Gebruikte programma's	107

Voorwoord

Er zijn zeer veel computerhobbyisten. Dit wordt duidelijk als we zien hoe veel mensen lid zijn van een club voor hobbycomputergebruikers.

Dat is niet verwonderlijk, want het werken met de computer en vooral het programmeren daarvan is een fascinerende bezigheid. Het overgrote deel van deze programmeurs werkt met de BASIC programmeertaal. Iedereen die zich daar enige tijd mee bezig heeft gehouden weet dat BASIC (en niet alleen BASIC) zijn beperkingen heeft. Deze zijn vaak gelegen in de relatief grote tijd die nodig is bij het 'vertalen' van BASIC naar machinetaal en velen zullen zich gerealiseerd hebben dat het in die gevallen beter is om direct maar in machinetaal te programmeren. Het tijdrovende vertalen wordt dan vermeden. Machinetaalprogrammeren moet echter ook worden geleerd. Nu is het de bedoeling van dit boek om u daarbij te helpen. Het leren doet u zelf, dit boek helpt u door de gegevens die u daarbij nodig hebt 'aan te geven', in de juiste volgorde en op een zo duidelijk mogelijke manier.

Ik mag wel aannemen dat u een Commodore 64 bezit. Daarmee hebt u dan een computer die, naar mijn mening, een zeer interessante microprocessor bezit. Deze processor vormt de kern van het eerste deel van dit boek. Behalve dat hierin aandacht is besteed aan de opbouw van de computer wordt u op de hoogte gebracht van de werking van alle instructies en adresseermethoden van de processor. Door deze instructies op de juiste wijze te combineren komt u tot een programma.

Voor het maken van een programma heeft u 'inzicht' nodig, inzicht in het probleem dat u moet oplossen. U kunt geen programma maken voor de beveiliging van uw huis als u niet weet op welke

manieren een ongenode gast naar binnen kan komen. Inzicht moet u ook hebben in het combineren van de instructies. Dit inzicht kunt u krijgen, gewoon door het veel te doen en een aldus gevormd programma op de computer uit te proberen. Dit inzicht kunt u ook krijgen door het bestuderen van bestaande programma's. Deze mogelijkheid biedt u het tweede gedeelte van dit boek met de meer dan vijftig kleinere en grotere programma's. Elk zo'n programma kan op de computer worden uitgetest. Uiteraard zijn de programma's eerst slechts klein, om later uit te groeien tot grotere. Elk programma heeft tevens tot doel u de gelegenheid te geven de mogelijkheden van de computer te leren kennen. Daarbij gaat het niet alleen om het toetsenbord en het scherm, maar ook om het normale gebruik van de randapparatuur met machinetaalprogramma's. Er wordt in dit boek zeer veel gebruik gemaakt van subroutines die in de BASIC interpreter en in het systeemprogramma aanwezig zijn. Van deze routines wordt u duidelijk getoond hoe ze moeten worden gebruikt.

De bedoeling is dat dit boek een op zich zelf staand geheel is. Het is een boek uit een serie over computers die door de uitgeverij De Muiderkring wordt uitgegeven. Wel wordt aangenomen dat u de mogelijkheden kent die uw computer u biedt bij het BASIC programmeren. Zeker is dat samen met het boek 'Leren programmeren met de Commodore 64', uit dezelfde serie, dit boek een zeer volledig beeld geeft van de mogelijkheden die de Commodore 64 heeft.

Ede, januari 1985

1. De opbouw van de computer

1.1. De samenstellende delen

Het meest centrale onderdeel van de computer is de *6510 processor*. Deze voert de opdrachten uit die aan de computer zijn gegeven in een programma.

Heeft u een BASIC programma in de computer gevoerd dan kan de processor de opdrachten die in dat programma worden gegeven, niet rechtstreeks uitvoeren. De opdrachten kunnen hem slechts worden toegediend in de vorm van codegetallen terwijl de soort van opdrachten die hij kan uitvoeren geheel verschillend zijn van de BASIC statements.

Dat wil zeggen dat een BASIC programma moet worden omgezet (vertaald) in een programma dat geschikt is voor de processor. Dit vertalen moet in de computer gebeuren door de processor zodat hiervoor een programma nodig is. Dit programma wordt de 'BASIC interpreter' genoemd. Uiteraard is de BASIC interpreter een programma dat de juiste vorm heeft voor de processor. Een dergelijk programma heet een *machinetaalprogramma*. Een direct in machinetaal geschreven programma is zeer veel sneller dan een BASIC programma omdat het vertalen achterwege kan blijven.

Een programma en ook de gegevens die voor dat programma nodig zijn, worden in de computer opgeslagen in een geheugen. Een geheugen bestaat uit registers die binaire getallen kunnen bevatten. De commodore 64 heeft geheugenregisters die binaire getallen kunnen bevatten van acht bits. Elk register is genummerd en zo'n nummer wordt een adres genoemd. Het laagste adres is 0, het volgende 1 enzovoorts. De computer kan 65536 adressen de baas, van 0 tot en met 65535. Een geheugenregister is een elektronische schakeling en voor het 'aanwijzen' van een geheugenplaats, het *adresseren*, zijn evenzoveel geleiders nodig als er bits nodig zijn voor het hoogste adres. Ook het adresseren geschiedt namelijk met binaire getallen. Er zijn daarom hiervoor zestien geleiders nodig ($2^{16} = 65536$). Deze zestien geleiders worden tezamen de *adresbus* genoemd. Voor het uitwisselen van de gegevens tussen het geadresseerde register en de processor zijn *acht* geleiders nodig, de *databus*.

Voor het inbrengen van de gegevens middels het toetsenbord, cassette recorder of floppy disk zijn *ingangspoorten* nodig. Hierop zijn de geleiders aangesloten die de verbinding vormen met het betreffende randapparaat (toetsenbord ed.) en de computer. Deze ingangspoorten zijn een bepaald soort elektronische schakelingen die als registers zijn te beschouwen. Is de verbinding van het randapparaat naar de computer spanningsvoerend (+5 V) dan is de ingangspoort '1', anders '0'. De ingangspoorten zijn samengevoegd tot registers van acht bits zodat op een 'poortregister' acht ingangslijnen kunnen worden aangesloten.

Voor het uitvoeren van de gegevens naar de randapparatuur (beeldscherm, floppy disk, cassette recorder en printer) zijn eveneens poortschakelingen nodig, de *uitgangspoorten*. Wordt een uitgangspoort hoog ('1') dan wordt een spanning van +5 V over de hierop aangesloten lijn naar het randapparaat gestuurd. De uitgangspoorten zijn als registers te beschouwen en zijn samengevoegd tot een eenheid van acht bits.

In de commodore 64 worden poorten toegepast die zodanig zijn ontworpen dat ze als in- en als uitgangspoort kunnen worden geschakeld, dit ter keuze van de gebruiker. Behalve het register met de poorten, dat wel het periferie dataregister wordt genoemd, is daarvoor ook nog een tweede acht bits register nodig voor het schakelen van de poorten als ingang of als uitgang. Dit is het Data Direction Register (DDR) en elk bit van dit register bedient één van de poorten. Wordt in bit 0 van het DDR een 1 geschreven dan wordt poort 0 van het poortregister een uitgang. Wordt in bit 0 van het DDR een 0 geschreven dan wordt poort 0 een ingang. Hetzelfde geldt voor de andere bits van het DDR en de overeenkomstige poorten van het poortregister (bit 1-poort 1, bit 2-poort 2 enz.). Zowel het periferie data register als het data direction register zijn in de geheugenruimte opgenomen en hebben dus een adres toegewezen gekregen. Een blokschematische voorstelling van het computersysteem geeft fig. 1.

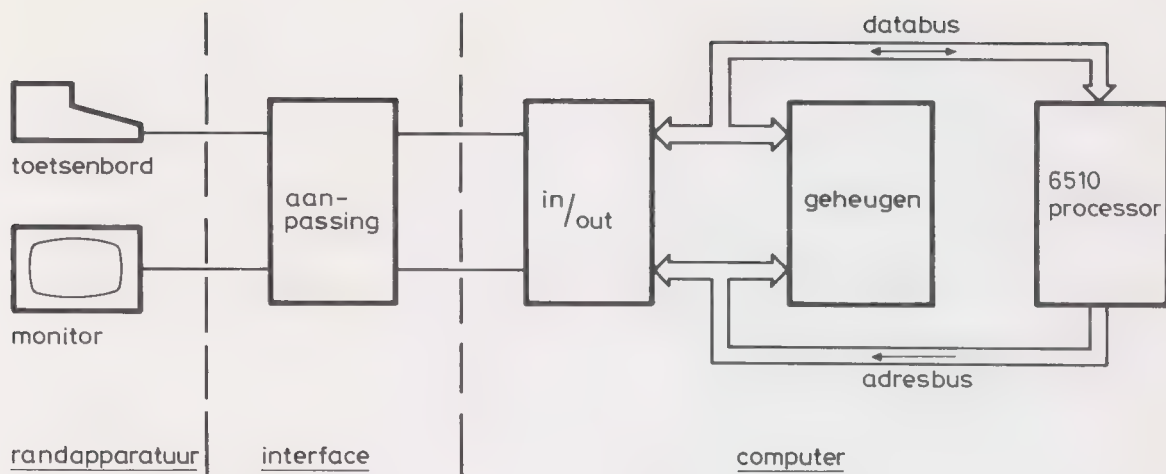


Fig. 1

Tot nu toe hebben we gedeelten van de geheugenruimte nodig gehad voor de BASIC interpreter en voor de in- en de uitgangsregisters. Om deze in- en uitgangsregisters te kunnen inschrijven en uitlezen en om de BASIC interpreter aan het werk te zetten is ook weer een programma nodig, het systeemprogramma, de 'KERNAL'. Dit is een programma dat in werking komt na het inschakelen van de computer en waardoor de diverse registers de juiste inhoud krijgen. Het zorgt voor de communicatie met de randapparatuur en bepaalt het bedieningsgemak van de computer. Zowel de BASIC interpreter als de KERNAL zijn opgeslagen in een zogenaamde READ ONLY MEMORY (ROM). Dit zijn geheugenelementen waarvan de inhoud van de registers niet verloren kan gaan bij het uitschakelen van de computer en wat niet kan worden ingeschreven, maar alleen kan worden uitgelezen. De overblijvende geheugenruimte is gevuld met geheugenelementen waarvan de registers zowel kunnen worden ingeschreven als uitgelezen, het RANDOM ACCESS MEMEORY (RAM).

De ROM geheugenelementen kunnen worden uitgeschakeld. In plaats daarvan worden automatisch RAM geheugenelementen ingeschakeld zodat de gehele geheugenruimte dan gevuld is met RAM geheugenelementen. Dit heeft dan de omvang van 64 Kbyte (1 Kbyte = 1024 byte, $64 \times 1024 = 65536$).

Behalve de databus en de adresbus zijn er nog een groot aantal verbindingen nodig voor het besturen van de computerelementen. Eén daarvan is de R/W lijn die aan de geheugenelementen doorgeeft of een getal moet worden ingeschreven ($R/W = 0$) of moet worden uitgelezen ($R/W = 1$).

1.2. Het hexadecimale codesysteem

De processor krijgt zijn instructies als digitale getallen. Deze instructies worden uit het geheugen opgehaald via de databus en zijn dus acht bits groot. Nu is het lastig om deze binaire getallen via het toetsenbord in te voeren in het geheugen, vandaar dat men een systeem hanteert waarbij de notatie van een getal veel eenvoudiger is. Dit is het hexadecimale codesysteem, een zestientallig stelsel. Hierbij worden de binaire getallen verdeeld in groepen van vier bits en elke groep wordt vervangen door een hexadecimaal cijfer. Met een groep van vier bits kunnen binaire getallen worden gevormd van 0000 tot en met 1111 (van 0 tot en met 15 decimaal). Dat betekent dat in het zestientallige stelsel steeds een cijfer is te vinden dat dezelfde waarde heeft als een bepaalde groep van vier bits. Lijst 1 geeft de notatie van de cijfers uit het hexadecimale stelsel met de binaire getallen van gelijke waarde.

Voor de zestien hexadecimale cijfers komen we aan de tien arabische tekens van 0 tot en met 9 te kort. Deze worden daarom aangevuld met de letters A tot en met F.

Een getal van acht bits wordt een byte genoemd. Een byte wordt in twee groepen van vier bits verdeeld (tetrades, Engels: Nibbles). Een byte kan daarom worden uitgedrukt in twee hexadecimale cijfers. Het getal 11010101 binair kan worden geschreven als \$ D5. Volgens lijst 1 komen de vier bits met de hoogste waarde (de hoge tetrade), 1101, overeen met \$ D en de vier bits met de laagste waarde, de lage tetrade, 0101, met \$ 5. Het teken \$ geeft aan dat een hexadecimaal getal bedoeld wordt.

Een geheugenadres is een binair getal van zestien bits (twee bytes) en kan in vier tetrades worden gesplitst. Een adres wordt daarom steeds met vier hexadecimal cijfers genoteerd. Voorbeeld:

1001101000100001 \triangleq \$ 9A21

Adressen en geheugeninhouden worden in dit boek hoofdzakelijk hexadecimaal gegeven. Waar geen verwarring mogelijk is zal het \$ teken worden weggelaten.

De geheugenruimte is in 'pagina's' verdeeld. Elke pagina omvat 256 adressen. De totale geheugenruimte omvat dan ook 256 pagina's ($256 \times 256 = 65536$). Het paginanummer wordt weergegeven door de hoge byte van het adres. Zo omvat pagina \$ 00 de adressen \$ 0000 tot en met \$ 00FF, pagina \$ 01 de adressen \$ 0100 tot en met \$ 01FF enzovoorts. Het hoogste paginanummer is \$ FF met de adressen \$ FF00 tot en met \$ FFFF.

Lijst 1. Hexadecimale getallen

Hexa-decimaal	binair	decimaal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

1.3. Indeling van het RAM geheugen

Het RAM geheugen kan worden ingedeeld naar het gebruik wat er van wordt gemaakt:

- Het data geheugen.
 - Het stapelgeheugen of Stack.
 - Het programma geheugen.
- Het datageheugen wordt gebruikt om gegevens in op te slaan die bij de verwerking van het programma moeten worden gebruikt. Deze gege-

vens zijn ingevoerde variabelen, tabellen en gegevens die sneller worden ingevoerd dan kunnen worden verwerkt (buffergeheugen).

- De Stack is het kladblaadje van de microcomputer. Hierin worden door de processor alle gegevens opgeslagen die tijdens het afwerken van het programma tijdelijk moeten worden bewaard. Dit geheugen ligt op pagina \$ 01.
- In het programmeergeheugen wordt in codevorm het programma opgeslagen. De volgorde van de instructies in het geheugen is dezelfde als die in het programma. Het adres van de eerste instructie is het startadres van het programma; alle andere instructies hebben een hoger adresnummer.

De volledige instructie bestaat uit de bewerking of handeling die de processor moet verrichten, de 'operatie', gevolgd door het getal waarop die operatie betrekking heeft, de 'operand', dan wel het adres waarop die operand te vinden is, meestal een adres van het datageheugen. De operatie kan alleen in de vorm van een codegetal in het geheugen geplaatst worden. Dit getal is de 'operatiecode'. Op de operatiecode volgt de operand dan wel het adres van de operand.

In fig. 2 is het stroomdiagram weergegeven van een gedeelte van een programma. Dit gedeelte telt de getallen \$ 15 en \$ 03 bij elkaar op. Het startadres is \$ 0200.



Fig. 2

Volgens de bij de 6510 gegeven tabel is de operatiecode voor de operatie 'haal op' \$ AD. Nu moet het adres volgen van de operand, want die staat in het datageheugen. Dit adres is 0032 en bestaat dus uit twee bytes: de byte met de hoogste plaatswaarde 00 (MSB, Most Significant Byte) en de byte met

de laagste plaatswaarde: (LSB, *Least Significant Byte*), 32. Een geheugenplaats kan slechts één byte bevatten zodat voor deze instructie in totaal drie geheugenplaatsen nodig zijn.

De operatiecode voor 'optellen' is \$ 69. Direct daarna volgt de operand. In figuur 3 is de situatie voor deze twee instructies in het geheugen geschetst. Let hierbij op de volgorde van de bytes van het adres van de eerste operand, eerst LSB en dan MSB. De computer werkt dit programmeel af door eerst op adres 0200 de operatiecode te lezen. Op de twee volgende adressen leest hij waar de operand te vinden is. Dan volgt de instructie voor het optellen van deze operand bij \$ 03.

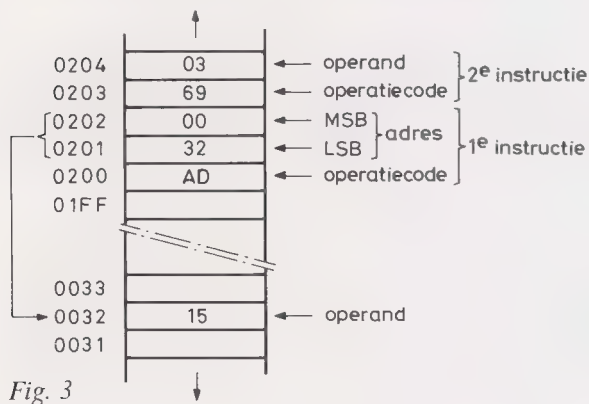


Fig. 3

1.4. Sprongopdrachten

Het geheugen van een microcomputer is opgebouwd uit registers van elk acht bits en elk register is een geheugenplaats. Dat betekent dat de instructie in 'code' in het geheugen moet worden gebracht (het geheugen wordt 'geladen' met de instructies). Ook het laden van het geheugen loopt via de processor.

Veronderstel nu dat voor een bepaald programma vijf geheugenplaatsen gevuld moeten worden met $00000000_{(2)} = 00_{(16)}$. De geheugenplaatsen zijn van 1 t/m 5 genummerd (gp1, gp2, ..., gp5). Het programma kan verlopen volgens fig. 4. De schrijfwijze voor: 'laad gp1 met \$ 00' is hier:

\$ 00 → gp1

(\$ 00 naar gp1).

Het getoonde programma kent vijf instructies, maar kan ook korter.

Bij het programma in fig. 5 wordt een bepaald register (bv. een geheugenplaats) aangeduid met x en gevuld met \$ 01.

Dit register noemt men een 'teller'. De volgende instructie laadt gp1 met \$ 00. Daarna wordt de teller met 1 'opgehoogd'. Het programma 'springt' nu terug naar de instructie die de aanduiding ('label') 'Loop' heeft gekregen. Dit is een sprongopdracht die gegeven moet worden ('Jump'), de computer doet niets uit zichzelf. De volgende instructie laadt gp2 met \$ 00, immers $x = 2$.

Deze kringloop gaat zo door zodat ook de volgende geheugenplaatsen worden geladen. De computer komt echter nooit tot een eind, want steeds wordt x opgehoogd en steeds wordt de kring doorlopen en een volgende geheugenplaats geladen.

Hier moet niet een gewone sprong worden toegepast maar een 'voorwaardelijke' sprong.

Een voorwaardelijke sprong ('Branch') betekent dat een sprong slechts optreedt (hier terug naar 'Loop') onder een bepaalde voorwaarde.

Bij het programma in fig. 6 wordt de teller geladen met \$ 05. De eerste geheugenplaats die nu geladen wordt is gp5 ($X = 5$). De teller wordt nu vermin-

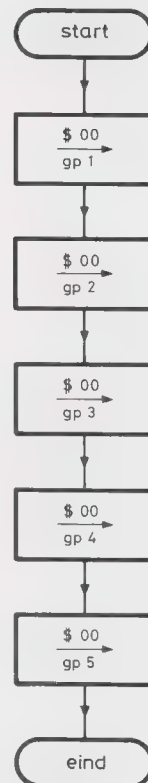


Fig. 4

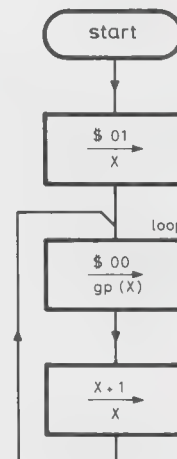


Fig. 5

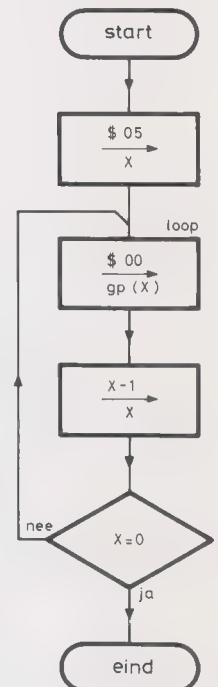


Fig. 6

derd met 1. Het volgende blok (de ruit) stelt nu de voorwaarde: als $X = 0$ volgt *geen* sprong. Zover is het nog niet. Het programma gaat terug naar 'Loop' en gp4 wordt geladen. Dit gaat zo door tot dat $x = 1$. Na het laden van de laatste geheugenplaats (gp1) wordt de teller opnieuw met 1 vermindert en is dus 0. Er wordt nu niet meer naar 'Loop' terug gesprongen en het programma is beëindigd.

1.5. Basisprogramma's voor logische functies

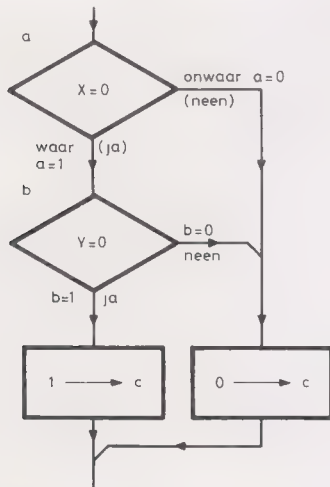


Fig. 7

De voorwaarde in het programma van fig. 6: $x=0$ kan ook als een uitspraak worden opgevat die dan waar (ja) of onwaar (neen) is. In fig. 7 is een programmadeel getekend waarin drie registers voorkomen. Van dit programmadeel is een waarheidstabel te maken waarin een 1 genoteerd wordt voor een *ware bewering* en een 0 voor een *onware bewering*. De 1 of de 0 heeft hier dus geen betrekking op de werkelijke inhoud van de registers x of y maar heeft betrekking op het al of niet waar zijn van de uitspraak over de inhoud van de registers. a is de uitspraak $x=0$, b is de uitspraak $y=0$. De kolom c geeft in tegenstelling met het voorafgaande *wel* de inhoud van het register aan. Er is hier dan ook sprake van een resultaat en niet van een uitspraak.

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Deze tabel is die voor de functie $c = a \wedge b$

Ook voor het programmadeel in fig. 8 is een waarheidstabel op te stellen:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

Deze tabel is die voor de functie $c = a \vee b$ zodat met het programmadeel in figuur 8 een 'of' functie is te verwezenlijken.

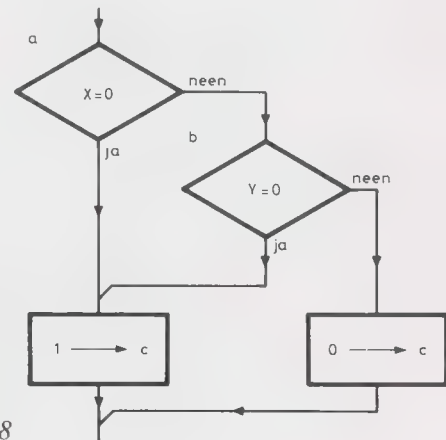


Fig. 8

Uiteraard is het ook mogelijk de functie $c = a \vee b$ te realiseren. Hiertoe dient het programmadeel in fig. 9. In dit figuur zijn twee punten gemerkt door een cirkel met een cijfer:

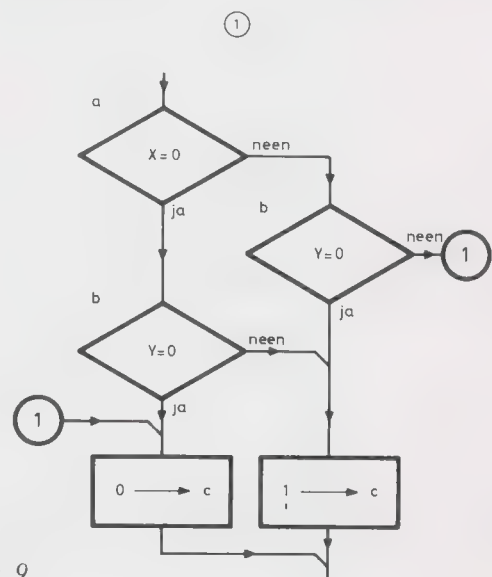


Fig. 9

Tussen de gelijk genummerde punten is een verbinding aanwezig. Deze methode wordt gevolgd om onduidelijkheden door kruisende lijnen e.d. te voorkomen.

Als een stroomdiagram groter is dan een bladzijde dan worden de verbindingen tussen de diverse programmadelen tussen de bladzijden met het volgende figuur aangegeven:



De waarheidstabel voor fig. 9 is:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Ook niet-functies zijn te realiseren. Het programmadeel in fig. 10 geeft de realisatie van de functie

$$c = a \wedge \overline{b}$$

a	b	$a \wedge \overline{b}$	c
0	0	0	1
0	1	0	1
1	0	1	1
1	1	1	0

Merk op dat het enige verschil met fig. 7 is dat de inhoud van register c is verwisseld.

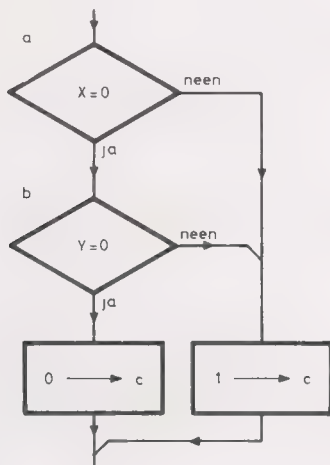


Fig. 10

Fig. 11 geeft de oplossing voor een samengestelde functie:

$$d = (a \wedge b) \vee c$$

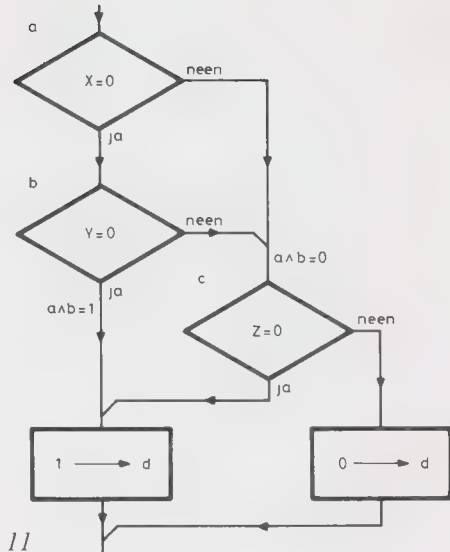


Fig. 11

a	b	c	$a \wedge b$	d
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

De waarheidstabel voor dit programmadeel kent drie variabelen. Er zijn dus $2^3=8$ mogelijkheden. Eerst wordt de tabel voor $a \wedge b$ gemaakt waarna een of-functie wordt gerealiseerd tussen deze en de tabel voor c.

Bij deze en ook bij vorige voorbeelden zijn steeds de uitspraken $x=0$, $y=0$ enz. genomen. Voor een voorwaardelijke sprong kan echter elke uitspraak genomen worden, zoals

$x = \text{positief};$
 $x = \text{negatief};$
 $x = n$ (n is een getal);
 $\text{carry} = 0;$
 $\text{carry} = 1;$
 enz.

1.6. De 6510 CPU

Voor het uitvoeren van de programma-instructies zijn er o.a. in de CPU een aantal registers aanwezig (fig. 12). Het register 'accu' wordt gebruikt bij het uitvoeren van de operaties en is een achtbits register. Ook de 'x'- en 'y'-indexregisters zijn acht-

bits. Zij kunnen beide bv. als teller worden gebruikt maar zijn eventueel ook nodig bij het adresseren.

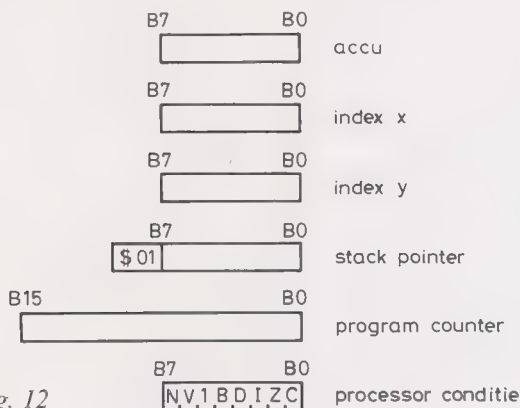


Fig. 12

Het register dat daarop volgt is de 'Stack pointer' S. Na het gebruiksklaar worden van de computer (bv. na het inschakelen van de voedingsspanning) worden alle bits van dit register '1'. Vullen we het aan tot 16 bits met \$ 01 dan is de totale inhoud dus \$ 01FF. De stack pointer wijst nu de geheugenplaats 01FF aan (fig. 13). Wordt door de processor deze geheugenplaats gevuld dan wordt tevens S met 1 verminderd. De inhoud van S wordt dan 01FE. Bij het vullen van deze plaats door de processor wordt de inhoud van S 01FD enz.

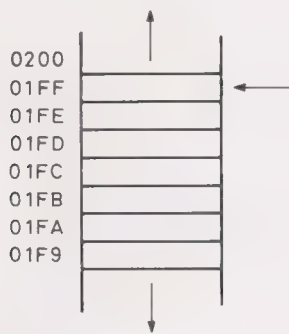


Fig. 13

Het laden van de Stack door de processor geschiedt van hoog naar laag terwijl het adres dat door de pointer wordt aangegeven steeds een *lege* geheugenplaats is. Het schrijven van de Stack gaat dus niet willekeurig maar: geheugenplaatsen voor geheugenplaats (vandaar: 'Stapelregister')

Ook het lezen gaat geheugenplaats voor geheugenplaats, te beginnen bij het laatst ingeschreven adres. Veronderstel dat het laatst ingeschreven

adres 01FB is, dan staat de Stack pointer op 01FA. Dit is dus een lege plaats. Bij het lezen wordt eerst de stackpointer met 1 verhoogd. Deze wijst dan de uit te lezen geheugenplaats aan (01FB). Nu wordt deze geheugenplaats gelezen. Voordat het volgende adres wordt gelezen wordt de Stack pointer weer met 1 verhoogd waarna het dan aangewezen adres wordt gelezen enz. De gelezen geheugenplaats kan als een lege geheugenplaats worden beschouwd (*alleen* bij de geheugenplaatsen van de Stack, niet die bij het programma-geheugen en ook meestal niet die van het datageheugen) zodat de Stack pointer ook nu een lege geheugenplaats aanwijst, de eerste onder de stapel.

Het Stack is een 'last in, first out' geheugen. Het getal dat het laatst is ingeschreven, wordt het eerst weer uitgelezen. De Stack pointer geeft altijd een adres aan op pagina \$ 01 en alleen de laagstwaardige byte hiervan kan veranderen.

De 'program counter' (PC) of 'instructieteller' wordt voor de start van een programma met het start-adres geladen. Dit register moet elk adres kunnen aangeven en omdat de adressen zestienbits zijn moet ook de PC zestienbits zijn. De teller zal tijdens het afwerken van het programma door de CPU steeds met 1 worden verhoogd en geeft dan steeds de geheugenplaats aan die door de CPU moet worden gelezen. Voor het uitvoeren van het programma worden zo de geheugenplaatsen stuk voor stuk na elkaar gelezen.

De teller kan tijdens een programma ook worden geladen met een geheel ander adres dan volgens de numerieke volgorde te verwachten was. Dit zien we dan gebeuren bij sprongopdrachten waarbij een programmeel (bv. een subroutine) moet worden doorlopen dat in een ander deel van het geheugen is ondergebracht. Ook zien we de teller wel een sprong terug maken zodat een reeds doorlopen programmeel opnieuw wordt doorlopen. Dit kan bv. het gevolg zijn van een voorwaardelijke sprongopdracht (fig. 6).

Het laatste register is het 'processor conditieregister' (P) of conditiecode-register. De plaatsen in dit register zijn gebruikt voor het Negatiebit N, het Overflowbit V, het Breakbit B, het Decimaalbit D, het interruptbit I, het Zerobit Z en het Carrybit C. Eén bit in dit register wordt niet gebruikt en zijn waarde is steeds '1'. Omdat deze bits een toestand van een resultaat van een bewerking kenmerken (het resultaat is 0, Z=1; het resultaat is negatief, N=1 enz.) worden ze ook wel 'flags' genoemd (Z-flag, N-flag enz.).

Een bijzonder bit is 'D' voor 'Decimal mode'. Als op deze plaats een '1' is geplaatst, zal de processor bij het rekenen automatisch de correcties toepassen die bij de BCD-code nodig zijn. In het geval dit een '0' is, wordt gewoon binair gerekend.

De meeste tijd dat een computer in werking is, zal worden 'verspild' met wachten totdat er iets gebeurt, bijvoorbeeld het wachten op het indrukken van een toets; in deze tijd mag de computer gerust een fractie van een seconde met iets anders worden belast.

Deze kleine tijd is genoeg om de computer een ander programma te laten doorlopen. Hiertoe wordt een 'signaal' gegeven, een 'interrupt'. Bij het ontvangen van een dergelijk signaal verlaat de computer dan het programma waar hij mee bezig is (het 'hoofdprogramma') en gaat dan het interrupt-programma doorlopen. Na het afwerken hiervan gaat hij weer verder met het hoofdprogramma.

Er kunnen bij het hoofdprogramma echter delen zijn die niet door een interrupt mogen worden onderbroken. In dat geval wordt in 'I' van het conditiecodelregister een '1' geplaatst. Het interruptsignaal wordt dan genegeerd en de computer blijft het hoofdprogramma afwerken. Pas als 'I' '0' is kan de computer weer op een interrupt reageren.

De plaats B in het conditiecodelregister wordt gebruikt om een '1' te noteren bij een 'Break'-instructie. Hierop zal nog worden ingegaan.

In fig. 14 is de interne organisatie van de processor geschetst. Behalve de in fig. 13 aangegeven registers zijn ook nog de Arithmetic and Logic Unit (ALU), het instructieregister, de decodering-, timing- en controlelogica, de data- en de adresbuffers weergegeven. De databuffers voeden de externe databus (de databus van de computer) en dienen tevens als ingangspoorten voor de processor. De adresbuffers voeden de externe adresbus.

De stuursignalen die nodig zijn voor de registers en de ALU worden geleverd door de decodering-, timing- en controle-unit. Deze signalen zorgen er voor dat de ALU de juiste rekenkundige of logische bewerkingen uitvoert en dat de juiste registers worden geladen met de gegevens van de interne databus van de processor, dan wel dat deze bus de inhoud van de registers leest.

De bewerkingen die de processor uitvoert volgen elkaar op 'met de regelmaat van een klok'. Hier-voor ontvangt de processor een impulsvormig signaal (clocksignal) en bij elke impuls voert de processor een bewerking uit.

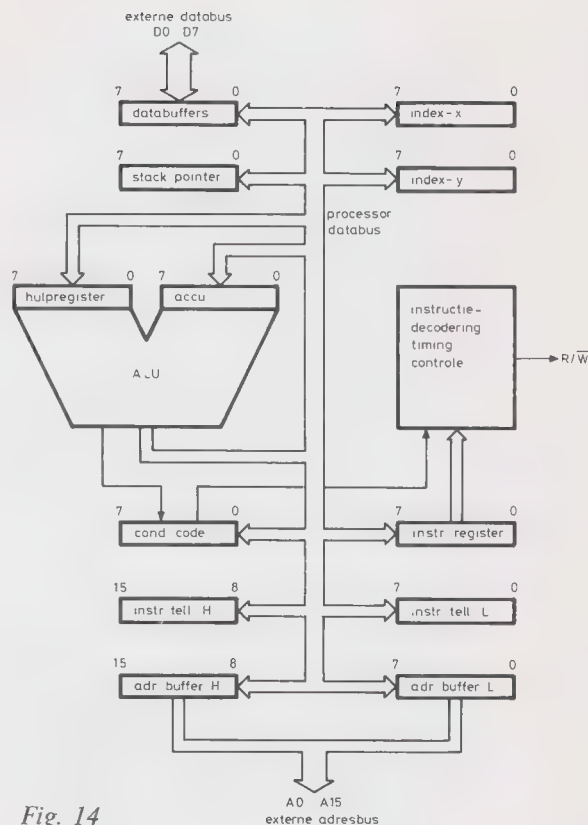
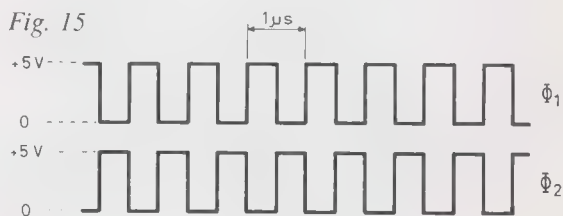


Fig. 14

Het kloksignaal is blokvormig, heeft een frequentie van 1 MHz en kent de fase Φ_1 en Φ_2 (fig. 15). De bewerkingen door de processor volgen elkaar daarom op met een herhaaltijd van 1 μ s.

De processor voert twee handelingen gelijktijdig uit. Eén handeling wordt de externe handeling genoemd. Tijdens een impuls van Φ_1 wordt het adres op de adresbus geplaatst. Tijdens de daaropvolgende impuls van Φ_2 komt de data van de geadresseerde geheugenplaats op de databus ($R/\bar{W}=1$) die dan eventueel door de processor kan worden geaccepteerd. Staat een operatiecode op de databus, d.w.z. de inhoud van de geadresseerde geheugenplaats is een operatiecode, en wordt deze geaccepteerd, dan spreekt men van een



'op-code fetch'. De operatiecode wordt dan in het instructieregister geplaatst om te worden gedecodeerd. Deze handeling heeft plaats gedurende één klokcyclus en duurt 1 μ s.

Gelijktijdig heeft een interne handeling plaats. Deze handeling hangt af van de data die door de processor in een vorige cyclus is opgenomen en kan bv. een rekenkundige of logische bewerking van de ALU zijn of het decoderen van een operatiecode. Tijdens deze cyclus kan eventueel intern de instructieteller worden opgehoogd.

Het gelijktijdig uitvoeren van meer handelingen heeft het voordeel dat de instructies in een zo kort mogelijke tijd kunnen worden uitgevoerd.

De rekenkundige en logische bewerkingen worden uitgevoerd door de ALU. Deze gebruikt daarbij twee registers, de accu en een hulpregister. Bij een bewerking wordt eerst het eerste getal in de accu geplaatst (fig. 16a). Hiervoor moet door de programmeur eerst de instructie 'load accu' in de vorm van een op-code (operatiecode) in het geheugen worden geplaatst. Na de op-code fetch door de processor en de decodering wordt de data vanuit de datapoorten via de processordatabus in de accu geladen.

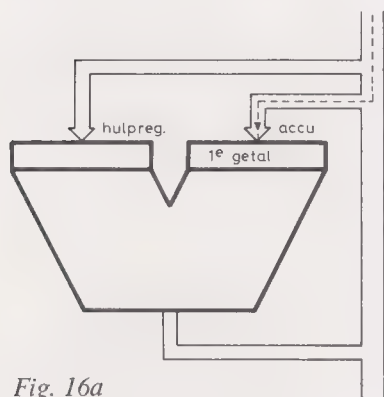


Fig. 16a

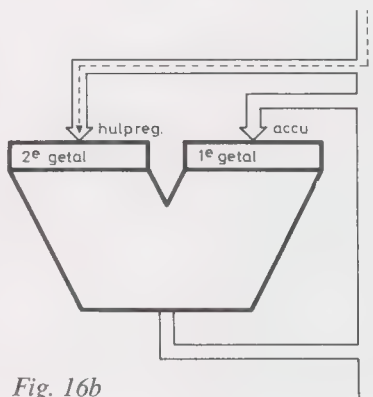


Fig. 16b

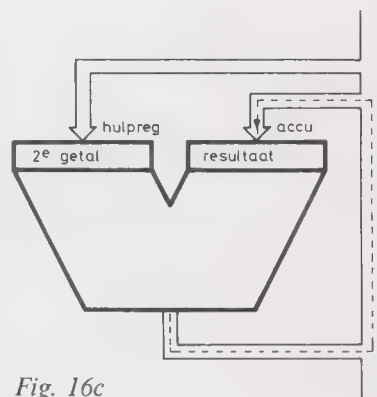


Fig. 16c

Het laden van het hulpregister vindt plaats na de fetch van de operatiecode van de bewerking die moet worden uitgevoerd (fig. 16b). De bewerking wordt nu door de ALU uitgevoerd en het resultaat wordt geladen in de accu. De oorspronkelijke inhoud van de accu (het 1e getal) gaat daarbij verloren (fig. 16c). Uiteindelijk is er weer een instructie nodig om de inhoud van de accu terug in het geheugen te plaatsen (store accu)

In totaal zijn drie instructies nodig voor het laten uitvoeren van een bewerking:

- 1e instructie: op code load load-accu + adres v.d. operand.
- 2e instructie: op-code bewerking + adres v.d. 2e operand.
- 3e instructie: op-code store-accu + adres v.d. geheugenplaats.

Op de chip van de 6510 processor bevinden zich nog twee extra registers. Deze zijn in de geheugenruimte opgenomen en zijn op gelijke wijze in te schrijven of uit te lezen als overeenkomstige registers van het geheugen. Het eerste register is een poortregister en bevindt zich op adres \$ 0001. Het tweede register is het data direction register van het poortregister en bevindt zich op het adres \$ 0000.

Er is reeds gesteld dat een lopend programma wel eens kan worden onderbroken door een interrupt. Een dergelijke interrupt kan tot stand worden gebracht met een signaal, waarvoor twee ingangen op de processor aanwezig zijn, namelijk de $\overline{\text{IRQ}}$ ingang en de NMI ingang. Normaal zijn deze ingangen hoog (+ 5V) en ze worden 'actief' door ze laag te maken (0 V).

Ten eerste de $\overline{\text{IRQ}}$ ingang (Interrupt Request). Wordt deze ingang laag dan wordt het lopende programma onderbroken, echter niet nadat de op dat moment lopende instructie geheel is afge maakt. Dan wordt naar een interruptprogramma gesprongen, tenzij bit I in het conditiecodelregister 1 is. In dat geval wordt het interrupt verzoek genegeerd. Anders is dat met de NMI (Non-Maskable Interrupt) ingang. Na het laag maken van deze in-

gang volgt steeds een interrupt, onafhankelijk van bit I. Na het doorlopen van het interruptprogramma wordt het hoofdprogramma weer vervolgd. Het is daarvoor nodig dat de gegevens van het hoofdprogramma bij het optreden van een interrupt niet verloren gaan. Na het laag maken van $\overline{\text{NMI}}$ of $\overline{\text{IRQ}}$ wordt door de processor eerst de lopende instructie afgemaakt. Daarna wordt de inhoud van de instructieteller en het conditiecodelogister naar de stack geschreven. De processor laadt nu de instructieteller met het adres van het interruptprogramma en doorloopt dit. Na afloop hiervan wordt het conditiecodelogister en de instructieteller weer met de oorspronkelijke gegevens uit de stack geladen zodat het hoofdprogramma kan worden voortgezet. Een andere belangrijke ingang van de processor is de reset ingang $\overline{\text{RES}}$. Ook deze

ingang wordt actief door hem laag te maken. Nu wordt echter, na het resetten van de processorregisters het systeemprogramma, de KERNAL, gestart. De $\overline{\text{RES}}$ ingang wordt dan ook gebruikt bij het inschakelen van de computer.

Noodzakelijk is dat de startadressen van de interruptprogramma's en van het systeemprogramma bekend zijn. De computer zoekt deze bij een $\overline{\text{NMI}}$ op de adressen \$ FFFA en \$ FFFB (in de volgorde lage byte — hoge byte, de $\overline{\text{NMI}}$ vector, bij een $\overline{\text{RES}}$ op de adressen \$ FFFC en FFFD, de reset vector en bij een $\overline{\text{IRQ}}$ op de adressen \$ FFFE en \$FFFF, de $\overline{\text{IRQ}}$ vector. De zes hoogste adressen van de geheugenruimte zijn dus voor deze vectoren gereserveerd. Uiteraard moet op deze adressen ROM geheugenelementen aanwezig zijn.

2. Instructies

2.1. De instructieset

De beschrijving van de instructies zijn voor het algemeen gebruik nog al onhandelbaar. Een van die instructies is b.v.:

Transfer index-X-register to accu.

Voor de instructies zijn dan ook afkortingen in gebruik, die zoveel mogelijk zodanig zijn gekozen dat de beschrijving er in terug te vinden is:

TXA: Transfer index-X-register to Accu

LDA: Load Accu.

De gebruikte afkorting wordt een 'Mnemonicsymbool' genoemd. Hiermee wordt een symbool bedoeld dat als geheugensteuntje voor de betekenis dient.

Het totaal aan instructies voor de CPU wordt de 'Instructieset' genoemd.

De instructies kunnen naar de aard van de bewerking worden ingedeeld in groepen:

- a. Transportinstructies.
- b. Arithmetische of rekeninstructies.
- c. Vergelijkingsinstructies.
- d. Logische instructies.
- e. Schuifinstructies.
- f. Voorwaardelijke spronginstructies.
- g. Onvoorwaardelijke spronginstructies.
- h. Overige.

2.2. Transportinstructies

Bij transportinstructies handelt het om datatransport tussen twee registers. Er zijn drie soorten transportinstructies:

Load-Store-instructies;

Transfer-instructies;

Push-Pull-instructies.

Een loadinstructie veroorzaakt een datatransport van een geheugenplaats naar een CPU-register. Een Store-instructie heeft het tegengestelde ten doel. Het datatransport is nu van CPU-register naar geheugenplaats.

Voor datatransport tussen CPU-registers onderling zijn transferinstructies. De laatste soort transport-

instructies zijn die voor datatransport van CPU naar Stack (Push) en van Stack naar CPU (Pull).

De eerste van de loadinstructies die we behandelen is:

LDA Load accu M→A

Deze instructie gaat steeds vooraf aan een rekenkundige of logische bewerking door de ALU. Het eerste getal wordt daarmee in de accu geladen (fig. 16.a). Het tweede getal wordt bij de bewerking in het hulpregister geplaatst. Zie hiervoor wat in hoofdstuk 1.6 over de ALU geschreven is.

Afhankelijk van de waarde van het getal dat in de betreffende geheugenplaats staat en dus van de waarde van het getal dat in de accu wordt geladen, worden N en Z van het conditiecodelregister 1 of 0. Is het getal negatief dan wordt N '1', anders '0'. Is het getal \$ 00 dan wordt Z '1', anders '0'. De instructie beïnvloedt dus de inhoud van N en van Z.

Het symbool dat in de regel

LDA Load accu M→A

op de uitdrukking 'Load accu' volgt, nl. M→A, geeft aan dat de inhoud van een geheugenplaats (M van Memory) in de accu wordt geladen: M gaat naar A.

Er zijn nog twee loadinstructies:

LDX Load index-x M→X

LDY Load index-y M→Y

Deze instructies zijn overeenkomstig met die voor LDA. Ook deze hebben invloed op N en Z van het conditiecodelregister en de inhoud van de geheugenplaats gaat niet verloren.

Een van de Store-instructies is

STA Store accu A→M

Na de bewerking door de ALU wordt het resultaat in de accu geplaatst (fig. 16.c). Indien hiermee geen verdere bewerkingen meer plaats moeten hebben, dient dit resultaat in het geheugen te wor-

den geplaatst. De instructie STA is hiervoor bestemd. Deze instructie heeft geen effect op het conditiecodelregister en de inhoud van de accu.

De twee volgende Store-instructies zijn:

STX Store index-x $X \rightarrow M$
 STY Store index-y $Y \rightarrow M$

Deze instructies zijn geheel overeenkomstig STA, ze hebben geen effect op het conditiecodelregister. De transferinstructies hebben datatransport tussen registers binnen de CPU tot gevolg.

TAX Transfer accu to index-x $A \rightarrow X$
 TAY Transfer accu to index-y $A \rightarrow Y$
 TXA Transfer index-x to accu $X \rightarrow A$
 TYA Transfer index-y to accu $Y \rightarrow A$
 TSX Transfer Stack pointer to index-x $S \rightarrow X$

Het transport van de gegevens is steeds van een register dat de 'bron' genoemd kan worden (Eng. Source) naar een ander register dat het 'doel' is (Eng. Destination). Welk register de bron en welk het doel is, volgt steeds duidelijk uit het mnemonicsymbool.

De bovenstaande transferinstructies hebben effect op de inhoud van N en Z, afhankelijk van de inhoud van de bronregisters. De inhoud van de bronregisters gaat niet verloren. Dit laatste geldt ook voor de instructie

TXS Transfer index-x to Stack pointer $X \rightarrow S$

Deze instructie heeft echter geen effect op het conditiecodelregister.

Het is reeds genoemd dat een interrupt een lopend programma onderbreekt. Hoewel de instructie waarmee de processor bezig is, wordt afge maakt voordat op een interrupt wordt gereageerd, kunnen de diverse registers in de processor gevuld zijn met data die niet verloren mag gaan. Dit zou bijv. kunnen zijn de stand van een teller (index-x of -y) die bij voortzetting van het hoofdprogramma nog aanwezig moet zijn. Maakt het interrupt-programma gebruik van deze registers dan gaat hun inhoud verloren en kan het hoofdprogramma niet meer op de juiste wijze worden voortgezet. Met de Push- (druk) instructies hebben we de mogelijkheid de registerinhouden in de Stack te bewaren:

PHA Push accu on Stack $A \rightarrow M_s$

Met M_s wordt hier een geheugenplaats in de Stack bedoeld. Deze instructie zet de inhoud van de accu in de Stack.

Veronderstel dat de instructie PHA (op-code \$ 48) op het adres 0200 in het geheugen staat. De geheugenplaats 0200 is dan gevuld met \$ 48. De verwerking van de instructie door de processor verloopt nu volgens de volgende tabel:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	Op-code PHA (\$ 48)	Fetch op-code	Voltooi vorige instructie. Verhoog PC tot 0201
2.	0201	Volgende op-code (\$ 8A)	Negeer de op-code	Decodeer de PHA- instructie. Houdt PC op 0201
3.	01FF	Accu	Store accu in Stack	Verlaag S tot 01FE
4.	0201	Volgende op-code (\$ 8A)	Fetch op- code	Verhoog PC tot 0202

Tijdens Φ_1 van de eerste klokcyclus komt 0200 op de adresbus te staan. Deze eerste cyclus is een leescyclus zodat de R/\overline{W} lijn '1' is en gedurende Φ_2 van deze cyclus de op-code van PHA (\$ 48) op de databus staat. Ook zal de op-code fetch plaats hebben. De op-code wordt nu in het instructieregister geladen. Dit zijn de *externe* handelingen van de processor. *Intern* heeft de voltooiing van de vorige instructie plaats. Laten we aannemen dat hierdoor de accu de inhoud 6A heeft gekregen.

Welke werking plaats heeft en of er een werking plaats heeft, is geheel afhankelijk van de aard van de vorige instructie. Ook nog intern wordt de instructieteller (PC) met 1 verhoogd tot 0201. Figuur 17.a geeft de inhouden van de registers van de processor weer. De stippellijn in deze tekening wil aangeven dat gedurende Φ_1 van de volgende klokcyclus de inhoud van de instructieteller naar de adresbuffers gaat en nu dus het adres 0201 krijgt aangewezen. Op dit adres staat de volgende op-code van het programma, hier \$ 8A. De tweede klokcyclus wordt echter door de processor geheel gebruikt voor het decoderen van de PHA-instructie en er kan daarom geen op-code fetch plaatsvinden. De inhoud van de registers verandert niet. Fig. 17.b geeft de situatie in de processor. Nadat de aard van de instructie is vastgesteld,

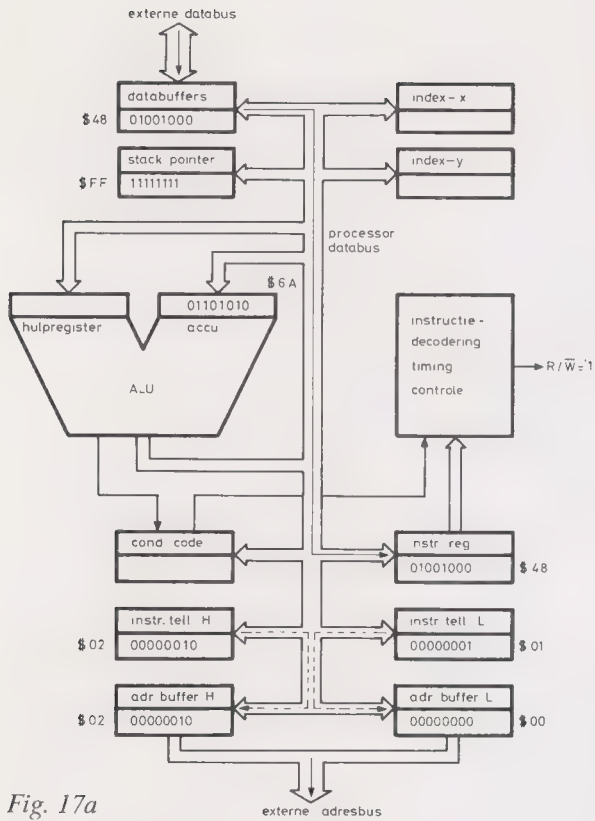


Fig. 17a

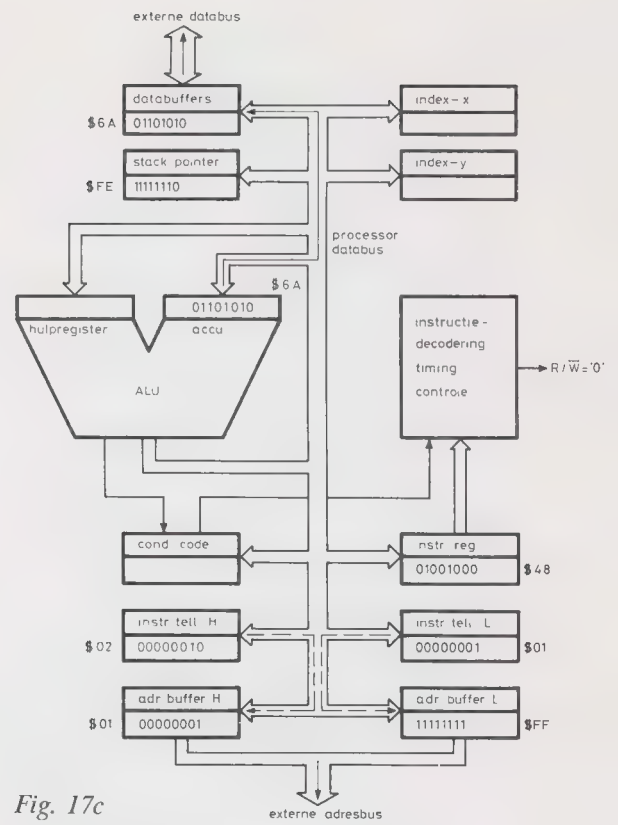


Fig. 17c

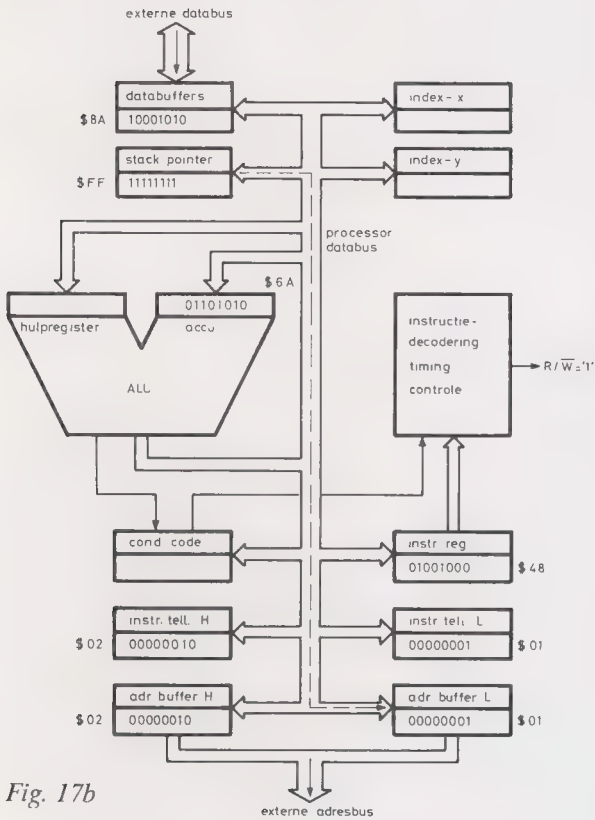


Fig. 17b

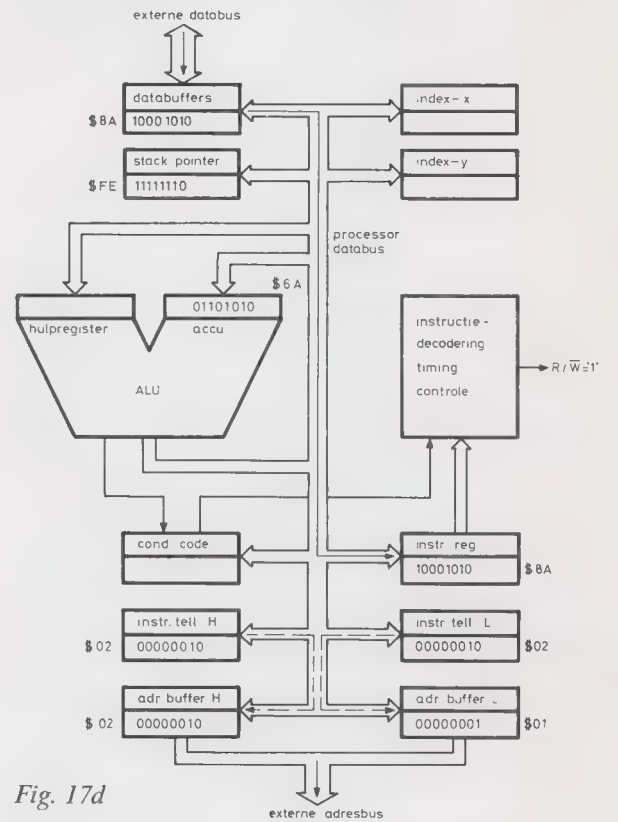


Fig. 17d

wordt in de derde klokcyclus gedurende Φ_1 de inhoud van de Stack pointer (S) in de adresbuffer L geladen en wordt de inhoud van de adresbuffer H gelijk aan \$ 01. De adresbus geeft nu het Stackadres 01FF aan en tijdens Φ_2 wordt de inhoud van de accu naar dat adres geschreven. De stuurlijn R/\bar{W} moet daarvoor '0' zijn. Intern wordt de Stack pointer verlaagd tot \$ FE (fig. 17.c).

De vierde cyclus is overeenkomstig de eerste. Ook nu heeft een op-code fetch plaats en deze cyclus behoort dan ook tot de volgende instructie. De instructie PHA heeft daarom voor de volledige uitvoering niet meer dan drie klokcycli nodig en duurt in totaal niet langer dan $3 \cdot 10^{-6}$ s (fig. 17.d).

Om ook het index-x of index-y-register in de Stack te krijgen, passen we een transferinstructie toe: TXA of TYA. Het betreffende indexregister is nu in de accu geladen en kan met PHA in de Stack geschreven worden.

Om de accu, het index-x en het index-y-register in de Stack te laden, dienen de volgende instructies elkaar op te volgen:

PHA accu in Stack
TXA
PHA index-x in Stack
TYA
PHA index-y in Stack

Dit zijn dan de instructies waarmee het interruptprogramma moet beginnen om de inhoud van de desbetreffende registers voor verloren gaan te behoeden. Ook het conditiecodelregister (P) kan naar de Stack:

PHP Push conditiecodelregister on Stack $P \rightarrow M_s$

Uiteraard wordt steeds na het laden in de Stack de Stack pointer met 1 verlaagd:

$S-1 \rightarrow S$

Aan het eind van het interruptprogramma moeten de registers van de processor weer in *omgekeerde volgorde* uit de Stack geladen worden door:

PLA Pull accu from Stack $M_s \rightarrow A$

Nemen we aan dat de instructie PLA op adres 0300 staat dan is de verwerking als volgt:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0300	op-code PLA	Fetch op-code	Voltooi vorige instructie. Verhoog PC tot 0301
2.	0301	Volgende op-code	Negeer de op-code	Decodeer de PLA instructie. Houdt de PC op 0301
3.	01FE	xxx	xxx	Verhoog S tot 01FF
4.	01FF	Inh. accu	Fetch inh. accu	S blijft 01FF
5.	0301	Volgende op-code	Fetch op-code	Laad accu

Na het decoderen van PLA in de tweede klokcyclus komt de inhoud van de Stack pointer op de adresbus. Deze geeft 01FE aan en dat is een 'lege geheugenplaats'. Hoewel het een leescyclus is, kan de processor met de data op de datalijnen (willekeurig) niets doen. Intern wordt de Stack pointer verhoogd tot 01FF. Dit adres komt in de vierde cyclus op de adresbus en op dit adres is de inhoud van de accu opgeslagen. De externe handeling accepteert de waarde van de databus.

In de vijfde cyclus komt de volgende op-code van adres 0301 op de databus. De externe handeling is de op-code fetch. Intern wordt de accu geladen met de uit de Stack gehaalde inhoud. Daar in de vijfde klokcyclus al de op-code fetch van de volgende instructies plaatsvindt, duurt de instructie PLA in totaal vier klokcycli.

Het laden van alle registers van de processor uit de Stack gaat met de volgende instructies:

PLA
TAY Index-y geladen uit Stack
PLA
TAX Index-x geladen uit Stack
PLA Accu geladen uit Stack

Let hier vooral op de volgorde in verband met 'Last in, first out'. Ook het conditiecodelregister kan uit de Stack geladen worden:

PLP Pull conditiecodelregister from Stack $M_s \rightarrow P$

Door de PLP-instructie zal eerst de Stack pointer worden verhoogd: $S+1 \rightarrow S$. Daarna wordt de inhoud terug in het conditiecodelregister geplaatst. Van de Push- en Pull-instructies heeft *alleen* PLA effect op N en Z van het conditiecodelregister.

Moet de inhoud van alle registers van de processor worden bewaard bij een interruptprogramma dan moet eerst de inhoud van het conditiecodelregister in de Stack worden geladen:

PHP
PHA
TXA
PHA
TYA
PHA

En weer terug aan het eind van het interruptprogramma:

PLA
TAY
PLA
TAX
PLA
PLP

Het conditiecodelregister wordt nu als laatste uit de Stack gehaald zodat zijn inhoud nu niet meer door de andere instructies gewijzigd kan worden (b.v. door een transferinstructie of door PLA).

2.3. De rekeninstructies

De arithmetische of rekeninstructies zijn in twee groepen te verdelen:

- Het optellen of aftrekken van twee getallen.
- Het verhogen of verlagen van een registerinhoud met 1.

Bij het optellen van twee getallen speelt het carry bit een belangrijke rol. Het carry bit wordt gebruikt als het resultaat van een optelling te groot is voor een register. De waarde van het carry bit moet dan ook worden gebruikt bij het optellen van twee getallen die groter zijn dan acht bits.

Voor het optellen is er slechts één instructie:

ADC Add memory to accu with carry
 $A + M + C \rightarrow A$

Deze bewerking heeft invloed op de waarde van N, Z, C en V van het conditiecodelregister:

N = 1. Het resultaat is negatief.
Z = 1. Het resultaat is nul.
C = 1. Het resultaat is te groot voor het register.
V = 1. De inhoud van N geeft de situatie niet juist weer.

In het volgende voorbeeld zijn twee positieve getallen van acht bits bij elkaar opgeteld:

	Binair	Decimaal
C	0	
a	00011011	27
M	01110111 +	119 +
a + M + C	10010010	146
C = 0 Z = 0 N = 1 V = 1		

Het achtste bit bij deze optelling is 1 (B7) vandaar N = 1, het resultaat is *niet* nul, dus Z = 0. Omdat het een optelling van twee positieve getallen betreft is N = 1 een onjuiste conditie, vandaar V = 1. Bij dit voorbeeld moet het carry bit vooraf 0 gemaakt worden. De inhoud van C is voor de optelling nl. willekeurig en zou daarom ook 1 kunnen zijn. Het carry bit wordt nul met de volgende instructie:

CLC Clear carry flag C = 0

Een optelling kan zowel binair als in de BCD-code gebeuren. De optelling van het voorbeeld is een binair optelling. De 'flag' D in het conditiecodelregister moet daarvoor nul zijn. Ook de inhoud van D kan willekeurig zijn en moet daarom nul worden gemaakt. Hiertoe dient de instructie:

CLD Clear decimal mode D = 0

De volgende instructies moeten elkaar opvolgen:

CLD
CLC
LDA, adres getal a.
ADC, adres M
STA, adres som.

Na CLD en CLC wordt eerst het getal a in de accu geladen (fig. 16a). Nu volgt de bewerkingsinstructie ADC. Deze instructie heeft tot gevolg dat getal M in het hulpregister geladen wordt (fig. 16b) en de optelling in de ALU tot stand komt waarvan

het resultaat in de accu komt te staan (fig. 16c). Het tweede getal is het getal M. De inhoud van de accu wordt met STA in de hiervoor bestemde geheugenplaats gebracht. Het optellen van twee getallen van zestien bits moet in twee stappen gebeuren daar de processor slechts getallen van acht bits kan optellen. In het volgende voorbeeld worden twee positieve getallen opgeteld

	hoog	laag
getal a	00111100	10100010
getal M	01010101	10111100+
som	10010010	01011110

C=0 Z=0 N=1 V=1

De optelling door de processor verloopt als volgt:

1 ^e stap: C	0	
a laag	10100010	
M laag	10111100+	
som laag	01011110	C=1 Z=0 N=0 V=1

De processor meent te doen te hebben met twee negatieve getallen, maar B7 van het resultaat is 0, vandaar V=1.

2 ^e stap: C	1	
a hoog	00111100	
M hoog	01010101+	
som hoog	10010010	C=0 Z=0 N=1 V=1
som	10010010	01011110

De inhoud van C, N en V van de tweede stap is bepalend voor de *gehele* som. Dit geldt niet voor Z, zoals blijkt uit het volgende voorbeeld waarin een negatief getal bij een positief is opgeteld.

getal a	00111100	10100010
getal M	11000011	10111100
som	00000000	01011110

C=1 Z=0 N=0 V=0

1 ^e stap: C	0	
a laag	10100010	
M laag	10111100+	
som laag	01011110	N=0 V=1 C=1 Z=0
2 ^e stap: C	1	
a hoog	00111100	
M hoog	11000011+	
som hoog	00000000	N=0 V=0 C=1 <u>Z=1</u>
som	00000000	01011110

Bij de tweede stap wordt Z '1'. Dit is niet correct voor wat betreft de gehele som. Als laatste een voorbeeld van het optellen van twee negatieve getallen.

getal a	11101101	11101011
getal M	11101110	11001110+
som	11011100	10111001

C=1 Z=0 N=1 V=0

1 ^e stap: C	0	
a laag	11101011	
M laag	11001110+	
som laag	10111001	N=1 V=0 C=1 Z=0
2 ^e stap: C	1	
a hoog	11101101	
M hoog	11101110+	
som hoog	11011100	C=1 Z=0 N=1 V=0
som	11011100	10111001

De volgende instructies zijn voor deze optellingen nodig:

	CLD
	CLC
1 ^e stap:	{ LDA, adres getal a laag, ADC, adres getal M laag, STA, adres som laag,
2 ^e stap:	{ LDA, adres getal a hoog, ADC, adres getal M hoog, STA, adres som hoog.

De instructie CLC komt alleen in de eerste stap voor. De waarde van C voor de tweede stap wordt bepaald door de eerste stap.
Het optellen in de BCD-code gaat geheel overeenkomstig het binair optellen. De nodige correcties worden automatisch gegeven als $D=1$. In plaats van CLD komt de instructie SED:

SED Set decimaal mode $D=1$

Verder zijn de instructies en de volgorde hiervan geheel gelijk aan die van het binair rekenen.
Bij het aftrekken wordt steeds het aftrektal in de accu geladen. De aftrekker is dan het getal M. De instructie die hierbij gebruikt wordt is:

SBC Subtract memory from accu with borrow
 $A-M-\bar{C} \rightarrow A$

De bewerking heeft effect op de inhoud van C, Z, N en V van het conditiecodelregister. De werking is te verklaren met het complementsysteem. Van het getal M wordt het twee-complement berekend. Hiertoe wordt dan M eerst de inverse bepaald en daarna 1 erbij opgeteld. Hiervoor wordt het carry-bit gebruikt dat daarom niet 0 moet zijn maar 1, vandaar

$A-M-\bar{C}$

Er volgen nu een aantal voorbeelden voor het aftrekken met getallen van acht bit lang. Eerst een aftrekking van twee positieve getallen met een positief resultaat.

getal a	01101001	getal M	01001100	
	C		1	
	a		01101001	
	\bar{M}		10110011+	
	<hr/>			
	a-M- \bar{C}	00011101	C=1 Z=0	
			N=0 V=0	

Aftrekking met twee positieve getallen met een negatief resultaat.

getal a	01001100	getal M	01101001	
	C		1	
	a		01001100	
	\bar{M}		10010110+	
	<hr/>			
	a-M- \bar{C}	11100011	C=0 Z=0	
			N=1 V=0	

Een positief getal aftrekken van een negatief

getal a	10110110	getal M	01011001	
	C		1	
	a		10110110	
	\bar{M}		10100110+	
	<hr/>			
	a-M- \bar{C}	01011101	C=1 Z=0	
			N=0 V=1	

In het resultaat is B7 nul, dus $N=0$. Het is een aftrekking waarvan het resultaat negatief is (een optelling van twee negatieve getallen) dus $V=1$.
Een negatief getal aftrekken van een positief

getal a	01011001	getal M	10110110	
	C		1	
	a		01011001	
	\bar{M}		01001001+	
	<hr/>			
	a-M- \bar{C}	10100011	C=0 Z=0	
			N=1 V=1	

De bewerking resulteert in een optelling van twee positieve getallen. $N=1$ ($B7=1$) vandaar $V=1$. Het resultaat is een positief getal.

Een negatief getal aftrekken van een negatief getal, met positief resultaat.

getal a	11010100	getal M	10010010	
	C		1	
	a		11010100	
	\bar{M}		01101101+	
	<hr/>			
	a-M- \bar{C}	01000010	C=1 Z=0	
			N=0 V=0	

Het programmeel dat voor de aftrekking nodig is, verloopt op gelijke wijze als dat voor optellen:

CLD,
SEC,
LDA, adres getal a,
SBC, adres getal M,
STA, adres verschil.

De tweede instructie is SEC in plaats van CLC. Het carrybit moet voor het rekenen in het twee-complement 1 zijn.

SEC Set carry flag $C=1$

Het aftrekken met twee getallen van zestien bits moet ook in twee stappen gebeuren.

getal a	01101010	00011101
getal M	01001011	01101011
C	1	
a laag	00011101	
\bar{M} laag	10010100+	
verschil laag	10110010	N=1 V=0 C=0 Z=0
C	0	
a hoog	01101010	
\bar{M} hoog	10110100+	
verschil hoog	00011110	N=0 V=0 C=1 Z=0
verschil	00011110	10110010

Het is niet mogelijk met SBC en SED decimale getallen af te trekken. Beter is het te rekenen in het tien-complement systeem.

De tweede groep van de rekeninstructies is voor het verhogen of het verlagen van een teller. Voor het op eenvoudige wijze ophogen van een teller met '1' gelden de volgende instructies:

INX	Increment index-x by one	$X + 1 \rightarrow X$
INY	Increment index-y by one	$Y + 1 \rightarrow Y$

Deze instructies beïnvloeden de inhoud van N en Z van het conditiecodelregister maar niet die van C en V. Dit is ook het geval met de instructie

INC	Increment memory by one	$M + 1 \rightarrow M$
-----	-------------------------	-----------------------

Deze instructie maakt het mogelijk om elke willekeurige geheugenplaats als teller te gebruiken. Dit is niet het geval met de vorige instructies. Deze hebben uitsluitend betrekking op het betreffende index-register.

Voor het verlagen van de tellers gelden de volgende instructies:

DEX	Decrement index-x by one	$X - 1 \rightarrow X$
DEY	Decrement index-y by one	$Y - 1 \rightarrow Y$
DEC	Decrement memory by one	$M - 1 \rightarrow M$

Ook nu wordt de inhoud van N en van Z beïnvloed. Echter niet de inhoud van C en V.

2.4. De vergelijkingsinstructies

De vergelijkingsinstructies zijn in principe rekeninstructies. Een vergelijkingsinstructie voert een aftrekking uit. De oorspronkelijke getallen gaan daarbij echter niet verloren omdat het resultaat niet in de accu geladen wordt. De grootte van het resultaat wordt dus niet bekend gemaakt. Het enige effect van de instructie is het beïnvloeden van de inhouden van N, Z en C van het conditiecodelregister. De vergelijkingsinstructies worden gebruikt voor het stellen van een *voorwaarde* bij een voorwaardelijke sprongopdracht. Er zijn daarbij drie mogelijkheden:

- Is kleiner dan.
- Is groter dan.
- Is gelijk aan.

De instructies die beschikbaar zijn, hebben betrekking op de accu, het index-x- en het index-y-register. Het tweede getal staat steeds in een geheugenplaats (M).

CMP	Compare memory and accu	A—M
CPX	Compare memory and index-x	X—M
CPY	Compare memory and index-y	Y—M

We onderzoeken de drie mogelijkheden:

Getal a is kleiner dan getal M, $a < M$.

De getallen a en M zijn beide positief.

$$a = 01011001 \quad M = 01101011$$

a	01011001	
\bar{M}	10010100	
C	1+	
	11101110	C=0, N=1, Z=0, V=0

Hetzelfde resultaat van de conditie-'flags' wordt verkregen als beide getallen negatief zijn.

De getallen a en M zijn tegengesteld van teken.

$$\begin{array}{ll} a \text{ is negatief} & M \text{ is positief} \\ a = 10010010 & M = 01010010 \end{array}$$

a	10010010	
\bar{M}	10101101	
C	1+	
	01000000	C=1, N=0, Z=0, V=1

$$a < M: C=0, N=1, Z=0, V=0 \text{ of (p)} \\ C=1, N=0, Z=0, V=1 \quad (q)$$

Zowel bij p als q is N ongelijk aan V zodat $N \nabla V = 1$

Getal a is groter dan getal M, $a > M$.
De getallen a en M zijn beide positief.

$$a = 01101011 \quad M = 01011001$$

$$\begin{array}{r} a \quad 01101011 \\ \overline{M} \quad 10100110 \\ C \quad \quad \quad 1+ \\ \hline 00010010 \end{array} \quad \begin{array}{l} C=1, N=0, \\ Z=0, V=0 \end{array}$$

Hetzelfde resultaat van de conditieflags wordt verkregen als beide getallen negatief zijn.
De getallen a en M zijn tegengesteld van teken.

$$\begin{array}{ll} a \text{ is positief} & M \text{ is negatief} \\ a = 01011001 & M = 10110110 \end{array}$$

$$\begin{array}{r} a \quad 01011001 \\ \overline{M} \quad 01001001 \\ C \quad \quad \quad 1+ \\ \hline 10100011 \end{array} \quad \begin{array}{l} C=0, N=1, \\ Z=0, V=1 \end{array}$$

$$a > M: C=1, N=0, Z=0, V=0 \text{ of (r)} \\ C=0, N=1, Z=0, V=1 \quad (s)$$

Zowel bij r als bij s geldt N is gelijk aan V dus $N \nabla V = 0$

Getal a is gelijk aan getal M.

$$\begin{array}{r} a = M = 01101011 \\ a \quad 01101011 \\ \overline{M} \quad 10010100 \\ C \quad \quad \quad 1+ \\ \hline 00000000 \end{array} \quad \begin{array}{l} C=1, N=0, \\ Z=1, V=0 \end{array}$$

Ook in dit geval is N gelijk aan V zodat $N \nabla V = 0$.

Voor de getallen a en M zijn vijf mogelijkheden:

$$a > M, a \geq M, a = M, a \leq M, a < M$$

In verband met de conditieflags zijn de volgende voorwaarden te stellen:

$$\begin{array}{ll} a > M: & Z \vee (N \nabla V) = 0 \\ a \geq M: & N \nabla V = 0 \\ a = M: & Z = 1 \\ a \leq M: & Z \vee (N \nabla V) = 1 \\ a < M: & N \nabla V = 1 \end{array}$$

2.5. Logische instructies

Er zijn drie logische bewerkingen die de processor kan uitvoeren, de 'EN' (\wedge) de 'OF' (\vee) en de 'Exclusief OF' (∇) bewerking. De instructies zijn:

$$\begin{array}{lll} \text{AND} & \text{And memory with accu} & A \wedge M \rightarrow A \\ \text{ORA} & \text{Or memory with accu} & A \vee M \rightarrow A \\ \text{EOR} & \text{Exclusief-or memory with accu} & A \nabla M \rightarrow A \end{array}$$

De instructies hebben effect op N en op Z van het conditiecoderegister. Het resultaat komt in de accu.

De computer past de instructies toe op twee overeenkomstige bits en werkt zo de woorden bit voor bit af. De voorbeelden van de logische bewerkingen in hoofdstuk 1.5 zijn volledig toepasbaar. Het is ook mogelijk met maskeren de waarde van een bit te bepalen, een tetrade te isoleren, tetrades samen te voegen en de inverse van een getal te bepalen. Stel dat de waarde van de derde bit van een getal bepaald moet worden.

$$\begin{array}{ll} a & 01001010 \\ \text{masker } b & 00000100 \\ \hline a \wedge b & 00000000 \quad Z=1 \end{array}$$

De instructievolgorde hiervoor moet zijn:

LDA \$ 04,
AND, adres getal a,
Bepaal inhoud van Z.

Door de eerste instructie wordt de accu geladen met het masker:

$$\$ 04 = 00000100_{(2)}$$

Door de tweede instructie voert de processor de 'EN' bewerking uit tussen het masker in de accu en het getal a. Het resultaat komt in de accu. Daarna dient de inhoud van Z van het conditiecoderegister te worden bepaald.

Door het plaatsen van het resultaat in de accu is het masker verloren gegaan. Dit kan nadelig zijn indien voor meer getallen de waarde van het derde bit bepaald moet worden. De instructiereeks moet dan zijn:

```
LDA $ 04,
AND, adres getal a,
Bepaal de inhoud van Z,
LDA $ 04,
AND, adres getal b,
Bepaal de inhoud van Z, enz.
```

De instructie LDA \$ 04 moet nu steeds worden herhaald.

Er is een instructie die de processor de 'En' bewerking laat uitvoeren zonder dat de inhoud van de accu verandert, de BIT-instructie.

BIT Test bits in memory with accu $A \wedge M$.

Met deze instructie wordt niet alleen de waarde van het gemaskeerde bit bepaald maar tegelijkertijd ook de waarde van B7 en B6. De instructie heeft een zodanig effect op het conditiecodelogister dat de waarde van N gelijk is aan de waarde van B7 van het onderzochte woord, de waarde van V gelijk is aan de waarde van B6 van het onderzochte woord en de waarde van Z afhankelijk is van de bewerking $A \wedge M$, zonder dat de inhoud van de accu daarbij verandert. Er wordt dus gelijktijdig de waarde van drie bits bepaald, die van B7, B6 en die van het gemaskeerde bit.

a	01001010	
masker b	00000100	
$a \wedge b$	00000000	N = 0 V = 1 Z = 1
a	10101101	
masker b	00000100	
$a \wedge b$	00000100	N = 1 V = 0 Z = 0

De instructiereeks voor het maskeren van meer getallen wordt nu:

```
LDA $ 04,
BIT, adres getal a,
Bepaal de inhoud van Z,
BIT, adres getal b,
Bepaal de inhoud van Z, enz.
```

Het steeds opnieuw laden van de accu met \$ 04 komt hier te vervallen.

Behalve dat het mogelijk is de waarde van een bepaald bit van een woord te bepalen is het ook mogelijk om een bepaald bit van een woord een waarde te geven. Stel dat van een woord het derde bit nul moet worden:

a	01011x01	hierin is x 0 of 1
masker M	11111011	
$a \wedge M$	01011001	

Omdat het derde bit van het masker 0 is en omdat de 'En' bewerking is toegepast wordt het derde bit 0.

De instructievolgorde is:

```
LDA, $ FB ($ FB = 11111011(2)),
AND, adres getal a,
STA, adres getal a.
```

Door de instructie LDA wordt het masker in de accu geladen. Na de AND-instructie is de inhoud van de accu gelijk aan het resultaat, dus *met* bit drie 0. Dan wordt dit resultaat geladen in de geheugenplaats van getal a. Hier vinden we dus het oorspronkelijke getal met bit drie 0.

Ook kan het bit de waarde 1 gegeven worden:

a	01011x01	x is 1 of 0
masker M	00000100	
$a \vee M$	01011101	

Hier is nu een OF-bewerking toegepast. De instructievolgorde wordt:

```
LDA, $ 04,
ORA, adres getal a,
STA, adres getal a.
```

2.6. De schuifinstructie

De schuifinstructies verplaatsen elk bit van een register één plaats naar rechts of naar links, afhankelijk van de betreffende instructie. Hierbij gaat geen enkel bit verloren. Ook het bit dat 'uit het register wordt geschoven' gaat niet verloren. Dit wordt steeds opgeslagen in C van het conditiecodelogister. Wordt aan één kant van het register een bit uitgeschoven, aan de andere kant wordt er een ingeschoven. Dit kan een '0' zijn maar ook de inhoud van C vóór het schuiven.

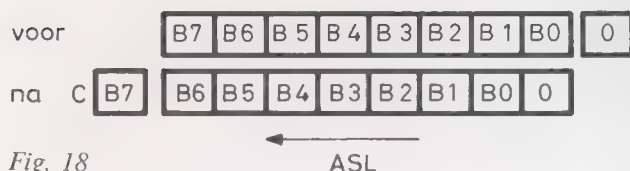


Fig. 18

Er zijn vier schuifinstructies:

- ROR Rotate right.
- ROL Rotate left.
- LSR Logical shift right.
- ASL Arithmetic shift left.

Fig. 18 geeft de situatie van een register voor en na een *shiftinstructie*. Bij ASL wordt B0 met '0' geladen en B7 komt in C terecht. Alle andere bits schuiven één plaats naar links op. Bij LSR wordt B7 met '0' geladen en B0 komt in C terecht. Deze instructies zijn elkaars tegengestelde. Het symbool voor ASL is



'Schuift men een binair getal in een register één plaats naar links dan wordt het met $2_{(10)}$ vermenigvuldigd.'

De bedoeling is dan wel dat de laagstwaardige bit (B0) '0' wordt. Dit is geheel in overeenstemming met de instructie ASL.

De instructie LSR is het tegengestelde van ASL en *deelt* door $2_{(10)}$.

'Schuift men een binair getal in een register één plaats naar rechts dan wordt het door $2_{(10)}$ gedeeld.'

Nu moet het hoogste bit (B7) '0' worden. Het symbool voor LSR is:

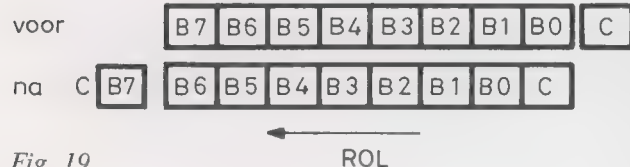
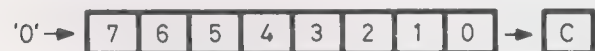


Fig. 19

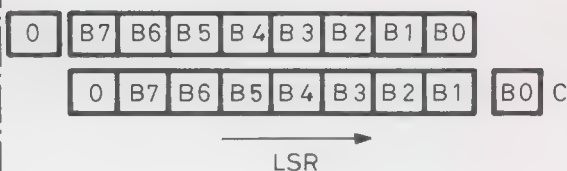
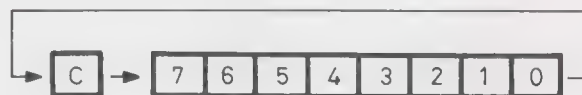
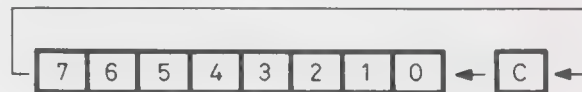


Fig. 19 geeft de situatie van een register voor en na een *Rotatie-instructie*. Zowel bij ROR als bij ROL is de carry in de keten opgenomen. De keten is 'gesloten' daar C eerst in het register wordt geschoven en later weer wordt gevuld uit het register. Het symbool voor ROL is:



En het symbool voor ROR:



2.7. Voorwaardelijke spronginstructies (Branch)

Indien de processor een programma aan het afwerken is, zal de instructieteller regelmatig worden verhoogd met 1 en zodoende steeds een adres aanwijzen dat 1 hoger is dan het vorige. Komt de processor in dit programma een spronginstructie tegen dan stopt hij met het steeds ophogen van de instructieteller en geeft hem een nieuwe inhoud, zodat met deze nieuwe inhoud als eerste adres een programmadeel doorlopen gaat worden dat in een ander deel van het geheugen is opgeslagen. Bij het doorlopen van dit programmadeel wordt de instructieteller weer steeds met 1 opgehoogd, zodat de adressen verder weer in numerieke volgorde worden afgewerkt.

Bij een voorwaardelijke spronginstructie treedt de sprong slechts dan op als aan een voorwaarde wordt voldaan. Wordt aan die voorwaarde niet voldaan dan wordt de instructieteller normaal verhoogd met 1 voor het volgende adres.

De voorwaarde voor de sprongen hebben steeds betrekking op de inhoud van een bit in het conditiecodelregister, zo in de vorm van:

'Maak een sprong als $Z = 1$ '

en ook

'Maak een sprong als $Z = 0$ '

Voor de voorwaardelijke spronginstructies bepalen de inhoud van C, Z, N en V of de sprong al of niet optreedt. Er zijn dan ook acht verschillende voorwaardelijke spronginstructies:

BCC	Branch on carry clear	Branch on $C = 0$
BCS	Branch on carry set	Branch on $C = 1$
BNE	Branch on result not equal	Branch on $Z = 0$
BEQ	Branch on result equal	Branch on $Z = 1$
BPL	Branch on result plus	Branch on $N = 0$
BMI	Branch on result minus	Branch on $N = 1$
BVC	Branch on overflow clear	Branch on $V = 0$
BVS	Branch on overflow set	Branch on $V = 1$

De Branchinstructies veranderen de inhoud van het conditiecodelregister niet. De operatiecode van de betreffende Branchinstructie moet gevolgd worden door een getal dat aangeeft hoeveel geheugenplaatsen naar voren dan wel terug moet worden gesprongen.

Een veel voorkomende voorwaardelijke sprong vinden we in fig. 6. Hierin wordt teruggesprongen naar 'Loop' als een teller (x) nog niet nul is. Er is echter geen Branchinstructie die de inhoud van een teller als voorwaarde kent. De Branchinstructie moet direct voorafgegaan worden door een bewerking die N, Z, C of V '0' of '1' maakt. Dit is in het genoemde voorbeeld inderdaad het geval.

Nemen we aan dat met x het index-x-register bedoeld is dan wordt de berekening:

$X-1 \rightarrow X$

tot stand gebracht met de DEX-instructie. Deze instructie zal Z '1' maken als, na een vermindering, de inhoud van het index-x-register nul is. Er moet naar 'Loop' gesprongen worden als het resultaat *niet* nul is, dus als $Z = 0$. Hiertoe dient de instructie

BNE

De laatste opdrachten van het programma uit fig. 6 moeten dan zijn:

DEX
BNE
n

Hierin is n het getal dat aangeeft hoeveel geheugenplaatsen verder dan wel terug moet worden gegaan. Voor een voorwaartse sprong is n een positief getal. Voor een achterwaartse sprong, zoals in dit voorbeeld, moet n een negatief getal zijn.

In het voorgaande is de voorwaarde van een sprong afhankelijk van het resultaat van een bewerking. Ook is het mogelijk de voorwaarde van een sprong afhankelijk te laten zijn van de grootte van een geheugeninhoud. Hierbij worden dan de vergelijkingsinstructies toegepast (CMP, CPX en CPY). Stel a is een getal in een register (bv. accu of indexregister) en M is een vergelijkingsgetal (bv. 5) dan zijn er vijf mogelijkheden als vergelijking:

$a > M$, $a \geq M$, $a = M$, $a \leq M$, $a < M$

Volgens hoofdstuk 2.4. hebben na een CMP-instructie (getal a in de accu en getal M in een geheugenplaats, CMP a-M) bij deze vijf mogelijkheden Z, N en V de volgende inhoud:

$a > M$	$Z \vee (N \vee V) = 0$
$a \geq M$	$N \vee V = 0$
$a = M$	$Z = 1$
$a \leq M$	$Z \vee (N \vee V) = 1$
$a < M$	$N \vee V = 1$

Stel dat een sprong moet worden gemaakt onder de voorwaarde $a < M$. Voor de conditieflags geldt dan $N \vee V = 1$. De waarheidstabel hiervoor is:

N	V	$N \vee V$
0	0	0
0	1	1
1	0	1
1	1	0

Alléén de regel twee en drie voldoen aan de voorwaarde voor de sprong. In dat geval wordt een sprong gemaakt naar een (nieuw) programmadeel dat zich in een ander deel van het geheugen bevindt.

Het stroomdiagram voor het realiseren van de voorwaarde $N \vee V = 1$ is in fig. 20 gegeven. Dit stroomdiagram kent twee verticale takken. De voorwaarden ($N=0$, $V=1$) zijn zodanig gekozen dat de rechte tak aansluit op de instructie van het volgende programmeel (met label 'Door' van 'Doorgaan'). Dit is eenvoudiger bij het opstellen van de instructievolgorde. De linker tak eindigt met de sprong naar het nieuwe programmeel. Na de CMP-instructie volgt de voorwaarde $N=0$. Wordt hier niet aan voldaan dan volgt een sprong naar de instructie die de label 'Exor' heeft meegekregen. Dit is in de richting a ($N=1$). Wordt hier wel aan voldaan (richting b, $N=0$) dan volgt in de volgende instructie de voorwaarde $V=1$. In de richting e wordt hier niet aan voldaan en er heeft een sprong plaats naar 'Door' omdat $N=0$ én $V=0$ (regel 1 van de waarheidstabel). Wordt wel aan de voorwaarde $V=1$ voldaan (richting f) dan volgt ook een sprong. In deze richting geldt nl. $N=0$ én $V=1$ en er wordt dus aan de algemene sprongvoorwaarde voldaan (regel 2 van de waarheidstabel). Het sprongadres wordt hier 'Nieuw' genoemd.

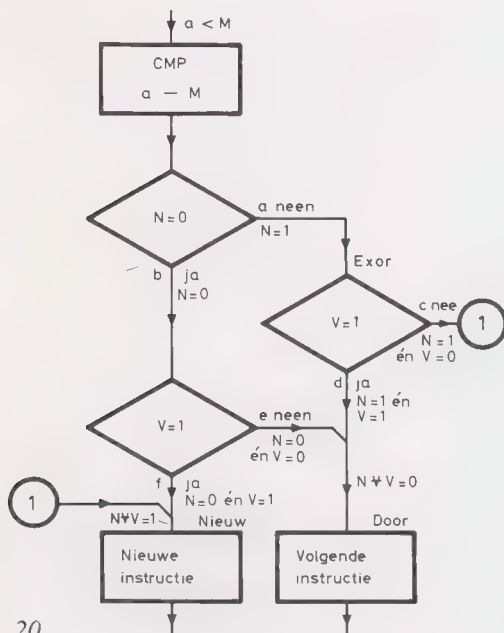


Fig. 20

Onder de label 'Exor' staat ook de voorwaarde $V=1$. In de richting c wordt hier niet aan voldaan zodat $N=1$ én $V=0$. Ook nu moet gesprongen worden naar 'Nieuw' (regel 3 van de waarheidstabel). Voor de richting d geldt $N=1$ én $V=1$ zodat

volgens regel 4 van de waarheidstabel geen sprong mag plaatshebben. Er moet nu gewoon doorgaan worden met het afwerken van het programma. De instructievolgorde geeft onderstaande lijst:

Label	Operatie adres
$a < M$	LDA, adres getal a CMP, adres getal M BMI, +4 Exor BVC, +4 Door BVS, +40 Nieuw
Exor	BVC, +38 Nieuw
Door	Volgende instructie

Bij het opstellen van deze lijst wordt eerst de linker verticale tak afgewerkt. De LDA-instructie brengt getal a in de accu. De daarop volgende CMP-instructie verandert de inhoud van de accu niet maar geeft wel de juiste waarde aan C, N, Z en V van het conditiecodelregister, afhankelijk van het resultaat van A-M. Nu moet een sprong volgen als $N=1$. Hiertoe is de instructie BMI. Het getal +4 achter deze instructie geeft aan dat vier plaatsen verder moet worden gesprongen (ook wel: vier plaatsen overslaan) naar de instructie voor 'Exor'.

Voor elke Branchinstructie zijn nl. twee geheugenplaatsen nodig. De volgende instructie veroorzaakt een sprong als $V=0$ naar 'Door'. Ook nu moeten vier plaatsen worden overgeslagen. Ook als $V=1$ moet een sprong gemaakt worden, nl. naar 'Nieuw'. Nu is voldaan aan de algemene voorwaarde $N \vee V = 1$. Er wordt nu 40 plaatsen vooruit gesprongen naar 'Nieuw', waar zich de nieuwe instructie bevindt. Het getal van 40 plaatsen is overigens geheel willekeurig gekozen.

De linker verticale tak is nu klaar en we gaan verder met de rechter verticale tak. Een sprong moet worden gemaakt als $V=0$, dus BVC. Ook nu moet naar 'Nieuw' worden gesprongen. We zijn met deze instructie al twee geheugenplaatsen in de richting van 'Nieuw' gekomen zodat na BVC het getal +38 komt. De volgende instructie is die met het label 'Door'. Het volgende programmeel sluit hierop dus gewoon aan.

In figuur 20 zijn twee takken te herkennen, één naar 'Nieuw' met als resultaat $N \vee V = 1$ en één naar 'Door' met als resultaat $N \vee V = 0$. Hiervan maken we gebruik bij de sprongvoorwaarde $Z \vee (N \vee V) = 1$ ($a \leq M$). De waarheidstabel hiervoor is:

Met 0300 op de adresbus staat op de databus de JMP-op-code. De adresbus gaat naar 0301 en vindt daar de laagstwaardige byte van het nieuwe adres. Op 0302 staat de hoogstwaardige byte. Vervolgens vinden we het nieuwe adres op de adresbus en op de databus de op-code die op dit adres is opgeslagen.

De tweede jumpinstructie is die van de subroutine:

JSR Jump to new location, saving return address (Jump to subroutine).

Het verschil met de vorige Jump is dat de stand van de instructieteller bij het verlaten van het hoofdprogramma wordt opgeslagen in de Stack. In tegenstelling tot de vorige Jump wordt na het doorlopen van de subroutine steeds weer teruggegaan naar het adres waar het hoofdprogramma verlaten was. Dit adres moet dus worden bewaard om het later weer te kunnen laden in de instructieteller. Na het doorlopen van de subroutine wordt de inhoud van de instructieteller weer uit de Stack gehaald en op de adresbus geplaatst zodat met het hoofdprogramma verder gegaan kan worden op de plaats waar het verlaten was.

Veronderstel dat de JSR-op-code (20) op adres 0300 staat. Op 0301 en 0302 staat dan in de volgorde *laag-hoog* het startadres van de subroutine (bv. 027A).

De inhoud van de betrokken geheugenplaatsen van het programmeergeheugen en van de Stack zijn weergegeven in fig. 22.

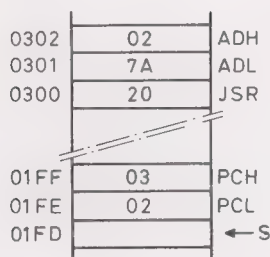


Fig. 22

Direct na het decoderen van de op-code wordt de stand van de instructieteller PC (0302) in de Stack geplaatst in de volgorde *hoog-laag* (PCH-PCL). De volgorde van de handelingen zijn weergegeven in bijgaand schema.

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0300	20 (op-code)	Fetch op-code	Maak vorige in- structie af. PC wordt 0301
2.	0301	7A (nieuw ADL)	Fetch nieuw ADL	Decodeer JSR. PC wordt 0302
3.	01FF	xxx	xxx	Store ADL
4.	01FF	03 (PCH)	Store PCH	Bewaar ADL S wordt 01FE
5.	01FE	02 (PCL)	Store PCL	Bewaar ADL S wordt 01FD
6.	0302	02 (nieuw ADH)	Fetch ADH	
7.	027A	op-code	Fetch op-code	ADL + 1 → PCL ADH → PCH

Na de op-code fetch in de eerste klokcyclus wordt in de tweede cyclus de fetch van de lage byte van het startadres (ADL) van het subroutineprogramma en het decoderen van de op-code uitgevoerd. Nu komt in de derde cyclus het Stackadres 01FF op de adresbus. Met dit adres gebeurt nog niets want de derde cyclus is nodig om ADL in de processor te laden en te bewaren. In de vierde cyclus wordt de hoge byte van de instructieteller in de Stack geladen op adres 01FF (store PCH) en in de vijfde cyclus de lage byte (PCL) op adres 01FE. De inhoud van de instructieteller gaat daarbij niet verloren. In de zesde cyclus komt de inhoud van de instructieteller weer op de adresbus (0302) en vindt de fetch van ADH van het subroutineprogramma plaats. Nu is het gehele adres van de subroutine in de processor. Dit adres wordt in de zevende cyclus op de adresbus geplaatst.

Aan het eind van de subroutine vinden we opnieuw een spronginstructie:

RTS Return from subroutine.

Deze instructie hoeft niet gevolgd te worden door een adres. Het adres waar het hoofdprogramma verlaten is, staat in de Stack. Het wordt uit de Stack gelezen en terug in de instructieteller geplaatst. Het voordeel hiervan is dat een subroutine op elke plaats in een programma kan worden aangeroepen.

De volgorde van de handelingen geeft de volgende tabel (RTS staat op adres 02A0).

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	02A0	60 (RTS)	Fetch op-code	Maak de vorige in- structie af. PC wordt 02A1
2.	02A1	volgende op-code	negeer op-code	Decodeer RTS
3.	01FD	xxx	xxx	S wordt 01FE
4.	01FE	02 (PCL)	Fetch PCL	S wordt 01FF
5.	01FF	03 (PCH)	Fetch PCH	
6.	0302	02	negeer de data	PC wordt 0303
7.	0303	volgende op-code	Fetch op-code	PC wordt 0304

Zoals uit de tabel blijkt, komt na het decoderen van de op-code RTS tijdens klokcyclus twee de inhoud van de Stack pointer S op de databus. Deze wijst nog een lege geheugenplaats aan zodat eerst S moet worden verhoogd tot 01FE in cyclus drie. In de vierde en de vijfde cyclus wordt de inhoud van de instructieteller PC uit de Stack gehaald, nu in de volgorde *laag-hoog*. Dit is geheel in overeenstemming met de werking van de Stack: last in, first out.

De instructieteller wordt geladen maar staat nog op een adres dat bij de laatste instructie van het hoofdprogramma hoort, voordat dit programma werd verlaten. De instructieteller moet eerst worden verhoogd voordat de op-code fetch kan plaatshebben.

Het is mogelijk om in het programma van een subroutine een tweede subroutine aan te roepen en eventueel hierin weer een derde enz. In fig. 23 staat in het hoofdprogramma op adres 0300 de instructie JSR, gevolgd door adres 027A als startadres van

een eerste subroutine. Het laatste adres van de instructieteller wordt in de Stack geladen. De Stack pointer komt op 01FD en geeft dit adres als lege geheugenplaats (figuur 24). Subroutine 1 start op 027A. Op adres 0280 van deze subroutine volgt JSR met aansluitend adres 02B0. Het laatste adres waarop de instructieteller staat is 0282 en dit wordt in de Stack geplaatst. De Stack pointer gaat naar 01FB. De tweede subroutine gaat van start op 02B0 totdat op 02C0 RTS (60) volgt. Het adres 0282 wordt uit de Stack gehaald en de Stack pointer gaat naar 01FD. De instructieteller wordt met 1 verhoogd tot 0283 en met de op-code van dit adres wordt de eerste subroutine verder afgemaakt. Ook nu volgt aan het eind van de subroutine (adres 02A0) RTS. De instructieteller wordt geladen met 0302 uit de Stack, de Stack pointer wordt 01FF en de instructieteller wordt verhoogd tot 0303, vanwaaruit het hoofdprogramma verder wordt afgewerkt.

Voor de goede orde: de adressen zijn volkomen willekeurig gekozen!



Fig. 24

De sprong naar een interruptprogramma verloopt ongeveer zoals de sprong naar een subroutine. De sprong kan 'Hardware' tot stand komen door NMI of IRQ 'laag' te maken. Aan het eind van het interruptprogramma staat (*moet* daar staan) de instructie:

RTI Return from interrupt

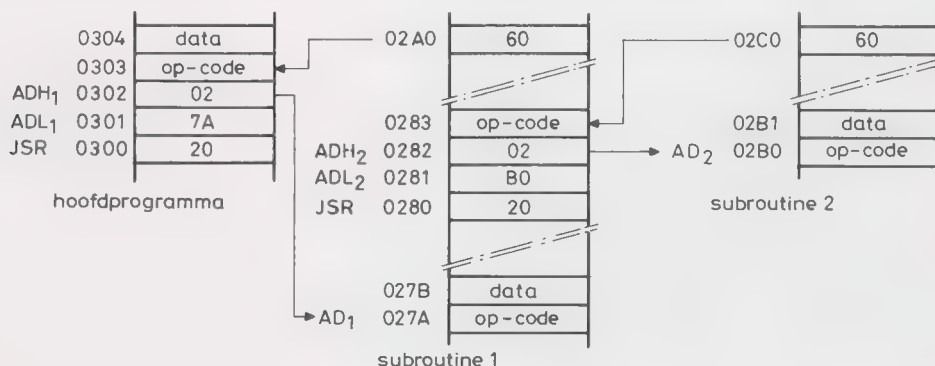


Fig. 23

Er heeft nu eenzelfde werking plaats als bij RTS. Omdat bij een interrupt niet alleen de instructieteller maar ook het conditiecodelregister in de Stack geplaatst is, moeten bij RTI uiteraard beide registers in de CPU worden geladen. Een interrupt kan ook 'Software' tot stand komen. Hiertoe dient de instructie:

BRK Forced Interrupt (Break command).

Komt de processor tijdens het doorlopen van een programma de operatiecode voor een BRK-instructie tegen (\$ 00), dan worden de adresbuffers geladen met het adres FFFE en één klokcyclus verder met FFFF. In deze geheugenplaatsen dient in de volgorde laag-hoog het startadres (vector) van een interruptprogramma te zijn geladen. Eerst worden echter door de processor die operaties verricht zoals die bij elke interrupt worden verricht, d.w.z. de instructieteller en het conditiecodelregister worden in de Stack geplaatst. Ook wordt bij deze instructie de B-flag gezet.

Dit is een bit in het conditiecodelregister dat slechts dan '1' wordt als de processor de BRK-instructie heeft ontvangen.

De BRK-instructie is niet te maskeren met de interruptflag. De instructie zal ook als I='1' een interrupt genereren. De handelingen van de processor bij een BRK-instructie geeft de volgende tabel:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0300	00 (BRK)	Fetch op-code	Voltooi laatste instructie. PC wordt 0301
2.	0301	Data	Negeer data	Decodeer BRK PC wordt 0302
3.	01FF	03 (PCH)	Store PCH	S wordt 01FE
4.	01FE	02 (PCL)	Store PCL	S wordt 01FD
5.	01FD	P	Store P	S wordt 01FC
6.	FFFE	7A Vector L	Fetch vector L	
7.	FFFF	02 Vector H	Fetch Vector H	7A gaat naar PCL
8.	027A	Op-code	Fetch op-code	027B gaat naar PC

In voorgaande tabel staat de BRK-op-code op 0300. De vector voor het interruptprogramma is 027A.

De verwerking van een 'Hardware'-interrupt, IRQ, verloopt in overeenstemming met deze tabel. In de eerste twee cycli wordt de PC echter niet verhoogd. Bij een IRQ en dezelfde adressen als in de voorgaande tabel zou dan als in houd van PC het adres 0300 in de Stack zijn geplaatst.

Dit geldt ook voor de interrupt NMI. In dit geval komen in de cycli zes en zeven resp. de vectores FFFA en FFFB op de adreslijnen.

Aan het eind van elk interruptprogramma moet de RTI-instructie (\$ 40) komen, om terug te kunnen springen naar het hoofdprogramma. De volgorde van de handelingen van de RTI op adres 02A0 zijn:

klok cyclus	adres bus	data bus	interne handeling	externe handeling
1.	02A0	40 (RTI)	Fetch RTI	Voltooi laatste instructie. PC wordt 02A1
2.	02A1	xxx	xxx	Decodeer RTI
3.	01FC	xxx	xxx	S wordt 01FD
4.	01FD	P	Fetch P	S wordt 01FE
5.	01FE	02 (PCL)	Fetch PCL	S wordt 01FF
6.	01FF	03 (PCH)	Fetch PCH	Laad instructieteller
7.	0302	op-code	Fetch op-code	PC wordt 0303

Op adres 02A1 is de data niet van belang voor de instructie. In klokcyclus drie wordt de Stack pointer verhoogd tot 01FD, het eerste adres van de Stack met de inhoud van het conditiecodelregister. Zoals dat bij de Stack noodzakelijk is, wordt eerst de Stack pointer verhoogd en daarna pas uitgelezen. Na het conditiecodelregister P wordt de instructieteller PC geladen zodat in de zevende klokcyclus de adreslijnen het adres 0302 aanwijzen.

Uit de voorbeelden volgt dat de instructieteller na RTI twee adressen hoger staat dan het adres van de BRK-instructie.

Dat is niet het geval bij een 'Hardware'-interrupt. Als de processor met een instructie bezig is en een interrupt optreedt dan maakt hij *eerst* de instructie af. De stand, die de teller heeft als de eerstvolgende op-code wordt geadresseerd, wordt in de Stack geladen, evenals de inhoud van het conditiecodelregister. Dit komt omdat bij een Hardware-interrupt gedurende de eerste twee klokcycli, zoals reeds is vermeld, de instructieteller niet wordt verhoogd. Na de RTI-instructie heeft het instructiere-

gister de stand die nodig is om de eerstvolgende op-code, waarmee het programma verder moet gaan, te lezen.

Zowel de Software-interrupt BRK als de Hardware-interrupt IRQ maken gebruik van dezelfde vectoren voor het interruptprogramma (de vectoren in de geheugenplaatsen FFFE en FFFF). Dit zou betekenen dat zowel bij de Hardware-interrupt IRQ als bij de Software-interrupt BRK hetzelfde interruptprogramma wordt gestart. Het is des ondanks mogelijk voor beide interrupts verschillende programma's te doorlopen.

In fig. 25 is de vector 0200 geladen in de adressen FFFE en FFFF. Op het adres dat deze vector aangeeft, start dus het interruptprogramma. De instructieteller en het conditiecodelregister zijn dan al in de Stack opgeslagen. De accu wordt geladen met het conditiecodelregister P door PLA. Dit register is nl. het laatste in de Stack opgeslagen. De Stack pointer S komt op 01FD. Deze moet echter naar 01FC om bij RTI de registers weer op de juiste wijze uit de Stack te krijgen. Hiertoe dient de instructie PHA. Omdat op adres 01FD al de inhoud van het conditiecodelregister is opgeslagen (bij het lezen verandert de inhoud van een register niet) zal de inhoud hiervan door de PHA-instructie niet veranderen. Wel verandert de inhoud van de Stack pointer. Deze wijst nu de lege plaats 01FC aan, zoals vóór de instructie PLA van adres 0200.

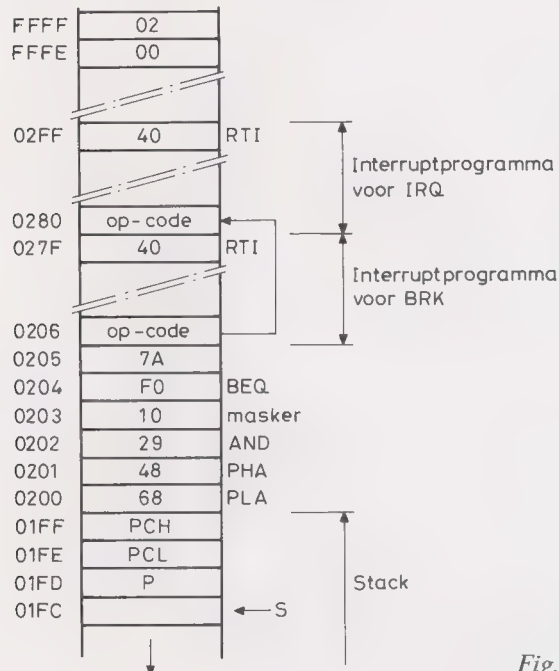


Fig. 25

Er volgt nu een AND-instructie met het masker \$ 10. Stel de inhoud van het conditiecodelregister was:

N	V	I	B	D	I	Z	C
1	0	1	0	0	0	0	0

Er heeft dus een Hardware-interrupt plaats gehad daar B='0'.

Hoewel I in het conditiecodelregister is gezet, is dit niet aan de inhoud van de Stack merkbaar. De I-flag wordt nl. in de zevende klokcyclus gezet na de reactie op de interrupt. De inhoud van het conditiecodelregister wordt al in de vijfde cyclus in de Stack opgeslagen zodat B3 van adres 01FD '0' is. Zou dit nl. niet het geval zijn geweest en B3 was '1' dan had niet op de IRQ kunnen worden gereageerd. De AND-bewerking wordt nu:

a	10100000	conditiecodelregister
M	00010000	Masker
$a \wedge M$	00000000	Z = 1

De BEQ-instructie op 0204 laat nu de instructieteller 122 adressen verder gaan:

$$122_{(10)} = 7A_{(16)} \text{ (Fig. 25)}$$

We belanden dan op adres 0280 waar het interruptprogramma voor IRQ start (de instructieteller wordt eerst opgehoogd tot 0206 en springt daarna naar 0280).

Was de interrupt ontstaan door een BRK-instructie dan zou de inhoud van het conditiecodelregister als volgt kunnen zijn:

N	V	I	B	D	I	Z	C
1	0	1	1	0	0	0	0

Dus: B='1'. De AND-bewerking is nu:

a	10110000	Conditiecodelregister
M	00010000	Masker
$A \wedge M$	00010000	Z = 0

Er heeft nu geen sprong plaats. De instructieteller wordt vanaf adres 0205 opgehoogd tot 0206 en start daar het BRK-interruptprogramma.

De Software-interrupt, BRK, wordt wel toegepast bij het 'debuggen' van een programma. Het is niet

gebruikelijk dat een ontworpen programma in één keer goed werkt. In het algemeen bevat het fouten en die zullen dan moeten worden opgespoord. De BRK-instructie is daarbij een hulpmiddel. Door in een programma een op-code te vervangen door de op-code van de BRK-instructie wordt het betreffende programma tot die BRK-instructie normaal doorlopen. Daarna wordt naar een interruptprogramma gesprongen dat ons in staat stelt bepaalde belangrijke registers (bv. de processorregisters) te controleren en zodoende vast te stellen of het programmadeel aan de verwachtingen heeft voldaan. We nemen als voorbeeld het programma van fig. 26 met als gegevens

a = 03
b = 05
c = 02

en gaan dan na hoe de verwerking moet verlopen.



Fig. 26

We vinden voor dit programma de volgende instructievolgorde:

CLD
CLC
LDA, adres getal a,
ADC, adres getal b,
SBC, adres getal c,
LSR

Als resultaat moet het getal in de accu gelijk zijn aan:

00000011 C=0

Bij een LSR van een *even* getal is C '0' en van een *oneven* is C '1', omdat bij een even getal B0 '0' en

bij een oneven getal B0 '1' is. Het antwoord moet hier dus zijn:

$$\frac{03 + 05 - 02}{2} = \frac{06}{2} = 03 \quad C=0$$

Bij controle van het antwoord vinden we echter 00000010 C=1

Er is dus een fout in het programma.

We brengen nu *in plaats van*, SBC, de instructie, BRK, in het programma:

CLD,
CLC,
LDA, adres getal a,
ADC, adres getal b,
BRK.

Bij controle na het doorlopen van dit programma is de inhoud van de accu

00001000 C=0

Dit is correct, daar

$$\begin{array}{r} C \quad 0 \\ A \quad 00000011 \\ b \quad 00000101 + \\ \hline A + b + C \rightarrow A \quad 00001000 \quad C=0 \end{array}$$

Hierin stelt A de accu voor waarin zich getal a en na de bewerking het resultaat bevindt. De BRK-instructie komt nu in de plaats van de instructie LSR:

CLD,
CLC,
LDA, adres getal a,
ADC, adres getal b,
SBC, adres getal c,
BRK.

Het resultaat in de accu is nu:

00000101 C=1

Dit keer is het resultaat niet correct. De fout zit dus in de aftrekking. De berekening die heeft plaatsgevonden is

$$\begin{array}{r} C \quad 0 \leftarrow \\ A \quad 00001000 \quad (\text{resultaat van ADC}) \\ \bar{c} \quad 11111101 + \\ \hline A - c - \bar{C} \rightarrow A \quad 00000101 \quad C=1 \end{array}$$

De berekening had moeten zijn:

$$\begin{array}{r} C \quad \quad \quad 1 \leftarrow \\ A \ 00001000 \\ \bar{c} \ 11111101 + \\ \hline A - c - \bar{C} \rightarrow A \ 00000110 \quad C = 1 \end{array}$$

De instructie, SBC, moet dus voorafgegaan worden door SEC. Het programma wordt:

CLD,
CLC,
LDA, adres getal a,
ADC, adres getal b,
SEC,
SBC, adres getal c,
LSR.

2.9. Overige

In deze groep zijn de instructies onder te brengen die invloed hebben op het conditiecodelregister.

Dit zijn de instructies:

CLC	CLI	CLD	CLV
SEC	SEI	SED	

De volgende instructies hiervan zijn nog niet genoemd:

CLI Clear interrupt disable bit.
SEI Set interrupt disable status.
CLV Clear overflow flag.

De instructies spreken voor zich en hebben geen nadere verklaring nodig. De laatste nog niet genoemde instructie is

NOP No operation.

Vooraf bij grote programma's kan het wenselijk zijn om op een aantal plaatsen enige ruimte open te houden om, als dat nodig is, bv. omdat het programma niet goed werkt, naderhand nog één of meer instructies tussen te kunnen voegen zonder het hele programma te moeten overschrijven. Het programma mag door die ruimte echter niet onderbroken worden. Er moet een aansluitende reeks van instructies blijven bestaan. Deze ruimtes kunnen nu worden opgevuld door de betreffende lege geheugenplaatsen te vullen met de instructie NOP. Deze heeft geen operatie tot gevolg maar zorgt ervoor dat een programma zonder onderbreking kan worden voortgezet. Ook kan de instructie worden gebruikt om de opengevallen geheugenplaatsen te vullen als achteraf een paar instructies komen te vervallen.

3. Niet geïndexeerde adresseermethoden

De titel van dit hoofdstuk: 'Niet geïndexeerde adresseermethoden' houdt gelijktijdig in dat er ook geïndexeerde adresseermethoden zijn. Het verschil tussen deze twee wordt duidelijk als de laatste in een volgend hoofdstuk wordt behandeld. We zullen ons dan ook maar niet wagen een definitie van deze methoden te geven.

Met een adresseermethode wordt de manier bedoeld die aangeeft waar in het geheugen de processor de operand vindt, waarmee de operatie moet worden uitgevoerd.

Het is niet alleen de 'omvang' van de instructieset (het aantal verschillende instructies) die de gebruiksmogelijkheden van de processor bepaalt, maar ook telt het aantal en de soort adresseermethoden mee.

De 6510 kent de volgende niet geïndexeerde adresseermethoden:

- a. Immediate.
- b. Zero page.
- c. Absolute.
- d. Indirect.
- e. Relative.
- f. Accu.
- g. Implied.

De geïndexeerde adresseermethoden zijn:

- a. Zero page, x.
- b. Zero page, y.
- c. Absolute, x.
- d. Absolute, y.
- e. (Indirect, x).
- f. (Indirect), y.

3.1. Immediate addressing

Bij deze methode bevindt zich de operand in het programmeergeheugen en wel direct na de operatiecode die betrekking heeft op die operand, bv.

Adres	Inhoud
0200	LDA
0201	\$ 04

Hier volgt de operand \$ 04 direct op de operatiecode. De totale instructie bestaat daarom uit twee bytes.

In het algemeen wordt deze adresseermethode gebruikt voor operanden die onveranderlijk zijn, zoals bv. een masker of de waarde waarmee een teller wordt geladen. De processor verricht de volgende handelingen:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	op-code (LDA)	Fetch op-code	Voltooi laatste instructie PC wordt 0201
2.	0201	data (04)	Fetch data	Decodeer op-code PC wordt 0202
3.	0202	op-code	Fetch op-code	Voer de LDA instructie uit. PC wordt 0203

Het betreft hier een op-code op adres 0200 met op 0201 de operand. De gehele instructie duurt twee kloktijden. In de derde cyclus wordt de gedecodeerde op-code uitgevoerd.

De instructies waarbij de immediate addressing kan worden toegepast, zijn:

LDA, LDX, LDY, ADC, SBC, AND, ORA, EOR, CMP, CPX, CPY.

3.2. Zero page addressing

Voor het dataregister wordt bij de 6510 in het algemeen de eerste pagina (pagina \$ 00) van het geheugen gebruikt. De reden hiervoor is de Zero page addressing mogelijkheid. Hierbij wordt de op-code van de instructie gevolgd door de laagstwaardige byte van een adres op pagina \$ 00 waarin de operand is opgeslagen. De hoogstwaardige byte hoeft niet te worden genoemd. Ten opzichte van het adresseren van alle andere pagina's waarvan het volledige adres gegeven moet worden, geeft deze methode een besparing van één geheugenplaats per instructie. Voorbeeld:

Adres	Inhoud
0200	LDA
0201	02

Ook nu moet de accu geladen worden met \$ 04. Deze operand staat echter op adres 0002. Achter de instructie LDA volgt nu \$ 02 dat het adres op pagina \$ 00 geeft waarop de data geladen is (fig. 27). De handelingen van de processor zijn:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	op-code (LDA)	Fetch op-code	Voltooi vorige in- structie. PC wordt 0201
2.	0201	ADL (02)	Fetch ADL	Decodeer LDA PC wordt 0202
3.	0002	data (04)	Fetch data	
4.	0202	op-code	Fetch op-code	Voer de LDA in- structie uit. PC wordt 0203

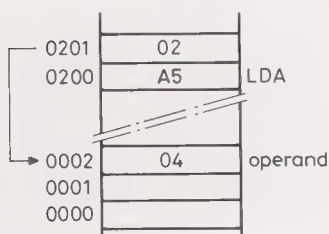


Fig. 27

In de derde cyclus wordt het adres 0002 op de adresbus geplaatst en de data wordt gelezen. In de vierde cyclus zijn we weer terug bij het programma op adres 0202, het adres van de volgende op-code. De instructie waarbij de zero page addressing kan worden toegepast, zijn:

LDA, LDX, LDY, STA, STX, STY, ADC, SBC, INC, DEC, AND, ORA, EOR, CMP, CPX, CPY, BIT, ASL, LSR, ROL en ROR.

3.3. Absolute addressing

Wordt een data in een ander deel van een geheugen geplaatst dan pagina \$ 00 dan moet de operatiecode die betrekking heeft op de operand worden gevolgd door het volledige adres van die operand. Voor het adres zijn dan twee bytes nodig. Direct na de op-code volgt de laagwaardige byte (ADL) en daarna de hoogstwaardige (ADH) van het adres.

In fig. 28 staat de operand \$ 04 op adres 037A. De op-code voor LDA (AD) staat op adres 0200. Op 0201 moet dan ADL (7A) volgen en op 0202 ADH (03).

Onderstaande tabel geeft de verwerking door de CPU:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	AD(LDA)	Fetch op-code	Voltooi voorgaande instructie. PC wordt 0201
2.	0201	7A (ADL)	Fetch ADL	Decodeer LDA. PC wordt 0202
3.	0202	03 (ADH)	Fetch ADH	PC wordt 0203
4.	037A	04 (data)	Fetch data	
5.	0203	nieuwe op-code	Fetch op-code	Voer instructie LDA uit. PC wordt 0204

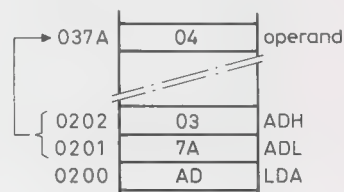


Fig. 28

Deze adresseermethode kan worden toegepast op de instructies:

LDA, LDX, LDY, STA, STX, STY, ADC, SBC, INC, DEC, AND, EOR, CMP, CPX, CPY, BIT, JMP, JSR, ASL, LSR, ROL en ROR.

3.4. Indirect addressing

Met de indirect addressing hebben we in principe al kennis gemaakt bij de interruptvectoren. In twee geheugenplaatsen staat een adres dat het beginadres is van een programma. Deze adressering voor de 'Hardware'-operaties kennen we voor de IRQ- en de NMI-interrupts en ook voor de Reset.

Ook de BRK-instructie maakt in principe van deze adressering gebruik. De vectoren staan hierbij in hiervoor vast aangewezen geheugenplaatsen. Dat is niet het geval bij de JMP-indirect-instructie. Hierbij wordt de plaats in het geheugen aangewezen waar de vector van het programmadeel staat waarnaar toe gesprongen moet worden. Hiervoor wordt de op-code gevolgd door resp. de laagstwaardige en de hoogstwaardige byte van een adres waar de laagstwaardige byte van de vector wordt gevonden.

In fig. 29 staat de op-code voor JMP op adres 0200. Daarna volgen ADL en ADH, die samen het

adres van vector L vormen. Eén adres verder vinden we echter vector H. Vector H en vector L leveren de plaats van het eerste adres van het program-madeel waarnaar toe moet worden gesprongen.

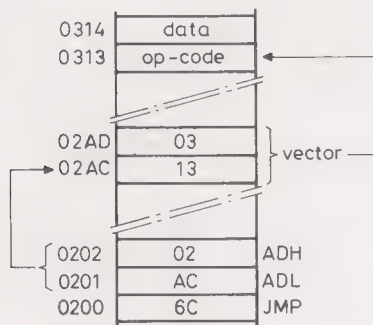


Fig. 29

De handelingen van de processor geeft onderstaande tabel.

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	6C (JMP)	Fetch op-code	Voltooi vorige in- structie. PC wordt 0201
2.	0201	AC (ADL)	Fetch ADL	Decodeer JMP. PC wordt 0202
3.	0202	02 (ADH)	Fetch ADH	Store ADL
4.	02AC	13 (vector L)	Fetch vector L	ADL wordt ADL + 1 (AD)
5.	02AD	03 (vector H)	Fetch vector H	Store vector L
6.	0313	nieuwe op- code	Fetch op-code	PC wordt 0314

Deze adressering wordt alleen toegepast op JMP.

3.5. Relative addressing

De relatieve adressering wordt uitsluitend toegepast bij de Branchinstructies. Na de op-code voor de Branch volgt, zoals reeds bekend is, een getal dat aangeeft hoeveel geheugenplaatsen verder dan wel terug moet worden gesprongen. Stel dat een Branchinstructie op adres 0200 staat en vier adressen verder moet worden gesprongen dan verricht de processor de volgende handelingen:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	op-code Branch	Fetch op-code	Voltooi vorige in- structie. PC wordt 0201
2.	0201	\$ 04	Fetch data	Decodeer Branchin- structie. PC wordt 0202
3.	0202	op-code	negeer op-code	Controleer de N, Z, V of C flag Vermeerder PC met \$ 04
4.	0206	nieuwe op-code	Fetch op-code	PC wordt 0207

Hier heeft een sprong plaats gehad. Was de betreffende conditieflag zodanig dat geen sprong mocht optreden dan zou in cyclus drie de op-code fetch hebben plaats gehad. Opgemerkt dient te worden dat voor de berekening van het nieuwe adres het aantal geheugenplaatsen van de sprong (hier \$ 04) wordt opgeteld bij de inhoud van de instructieteller als deze al op de nieuwe instructie staat (0202). In dit voorbeeld is de optelling dus $0202 + 0004 = 0206$.

In het algemeen is niet bekend hoeveel geheugenplaatsen de instructieteller moet overslaan (de grootte van de sprong). Wel is dan het adres bekend waarheen de sprong moet gaan. De berekening van de voorwaartse sprong verloopt als volgt: Stel, de op-code van de Branchinstructie bevindt zich op adres 02BA en gesprongen moet worden naar adres 030C. Eerst wordt het op-code-adres met twee verhoogd tot 02BC. De berekening wordt nu:

	Hex	Binair
sprongadres	030C	00000011 00001100
op-code-adres + 2	02BC	00000010 10111100
	0050	00000000 01010000

De volledige instructie wordt:

op-code
\$ 50

Ook bij een achterwaartse sprong moet het op-code-adres met twee worden verhoogd. Stel dat het op-code-adres 0325 is en het sprongadres 02D4. Eerst het op-code-adres met 2 verhogen $0325 + 0002 = 0327$

	Hex	Binaire
sprongadres	02D4	00000010 11010100
op-code-adres + 2	0327—	00000011 00100111—
	FFAD	11111111 10101101

De instructie is

op-code
\$ AD

De getallen die de grootte van een sprong aangeven zijn voor een voorwaartse sprong positief (\$ 01 t.m. \$ 7F) en voor een achterwaartse sprong negatief (\$ FF t.m. 80). De sprongen kunnen daarom niet groter zijn dan 127 geheugenplaatsen. Moet de sprong groter zijn dan moet naar een tussenadres worden gesprongen. Op dit adres staat dan een JMP-instructie naar het betreffende adres.

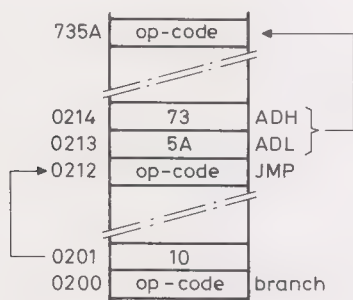


Fig. 30

In fig. 30 staat de Branchinstructie op 0200 en het sprongadres is 735A. Deze sprong is te groot voor een Branch en een JMP-instructie op 0212 zorgt voor de tussenstap.

3.6. De Accu en Implied addressing

Van de Accu en implied addressing hebben de instructies betrekking op de registers in de CPU en behoeven dus geen nadere adressering. Het zijn enkelbyte-instructies en worden toegepast op:

ASL, LSR, ROL, ROR voor de accu en verder TAX, TAY, TXA, TYA, TXS, TSX, PLA, PHA, PLP, INX, DEX, INY, DEY, CLC, CLD, CLI, CLV, SEC, SED, SEI, NOP, RTS, RTI en BRK.

Hoewel de BRK-instructie een enkelbyte-instructie is en als zodanig onder Implied addressing wordt genoemd, is de werking in principe die van de indirecte adressering. De handelingen van de processor voor de instructie, TAX, zijn:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	op-code (TAX)	Fetch op-code	Voltooi vorige instructie. PC wordt 0201
2.	0201	op-code	negeer op-code	Decodeer TAX
3.	0201	op-code	Fetch op-code	Voer instructie TAX uit. PC wordt 0202

4. Geïndexeerde adresseermethoden

4.1. Zero page index addressing

Bij de geïndexeerde adresseermethoden wordt de inhoud van het index-x- of het index-y-register gebruikt. Steeds wordt de inhoud van één van de indexregisters opgeteld bij een gegeven adres om het uiteindelijke adres van de operand te verkrijgen.

Fig. 31 demonstreert het nut van de geïndexeerde adressering. De eerste zes geheugenplaatsen in pagina \$ 00 moeten met \$ 00 worden geladen. Als teller wordt het index-x-register gebruikt. De inhoud van dat register is eveneens bepalend voor het adres dat moet worden geladen. Als *basis* wordt adres \$ 0002 aangewezen. Het uiteindelijke adres is dan 0002 + x. Het eerste adres dat wordt geladen is

0002 + 05 = 0007

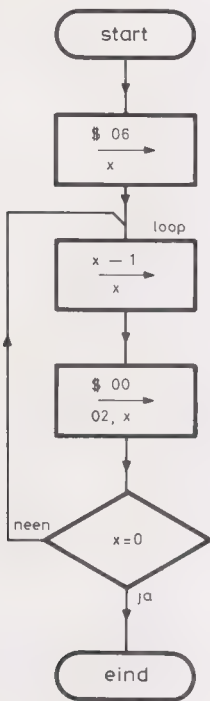


Fig. 31

Omdat de teller in elke cyclus met 1 wordt vermindert wordt daarna geheugenplaats 0006 geladen, in de volgende cyclus 0005 enz., tot de laatste ge-

heugenplaats 0002. De handelingen die de proces-sor bij zero page addressing verricht, zijn als volgt (op-code op 0205, x = 05):

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0205	95 (STA)	Fetch op-code	Voltooi vorige in- structie PC wordt 0206
2.	0206	02 (BAL)	Fetch Basis adr. L	Decodeer LDA. PC wordt 0207
3.	0002 (00, BAL)	xxx	xxx	ADL wordt BAL + x = 02 + 05 = 07
4.	007	Accu (00)	Store accu	
5.	0207	Op-code	Fetch op-code	PC wordt 0208

Het is niet mogelijk om op deze manier een geheu-genplaats op pagina \$ 01 te adresseren. Stel het ba-sisadres is 00E5 en de inhoud van het index-x-re-gister is \$ 20 dan zou volgens de optelling

00E5 + 20 = 0105

geheugenplaats 0105 geadresseerd moeten worden. De geadresseerde geheugenplaats wordt echter 0005, dus een adres op pagina \$ 00. Stel het pro-gramma van fig. 31 start op adres 0200. De instruc-tievolgorde toont de volgende tabel.

adres	label	mnemonic- symbool	adreseer- methode	geheugen- inhoud
0200		LDA	immediate	A9
0201		\$ 00		00
0202		LDX	immediate	A2
0203		\$ 06		06
0204	loop	DEX	implied	CA
0205		STA	zero page, x	95
0206		\$ 02		02
0207		BNE	relative	D0
0208		FB		FB
0209				

De laatste bewerking vóór de sprongopdracht die invloed heeft op het conditiecoderegister is DEX. Er moet naar 'loop' gesprongen worden als de in-houd van het index-x-register *niet* nul is, dus als

$Z=0$. Hiertoe dient de instructie BNE. De STA-instructie tussen DEX en BNE heeft geen invloed op het conditiecoderegister.

De berekening van de sprong verloopt als volgt:

	Hex	Binair
Sprongadres (loop)	04	00000100
Adres BNE + 2	09—	00001001—
	FB	11111011

De inhoud van geheugenplaats 0208 wordt FB.

De instructies waarbij zero page x adressering kan orden toegepast zijn:

LDA, LDY, STA, STY, ADC, SBC, INC, DEC, AND, ORA, EOR, CMP, ASL, LSR, ROL en ROR.

Op overeenkomstige wijze kan ook het index-y-register worden gebruikt voor zero page indexing, nu echter alléén voor de instructies LDX en STX.

4.2. Absolute index addressing

Bij absolute index addressing is het basisadres zoals bij absolute addressing een adres van twee bytes. Deze adresseerwijze wordt toegepast bij adressen op alle pagina's behalve bij pagina \$ 00. In tegenstelling tot de zero page indexing is het hierbij mogelijk om door indexing op een adres te komen in de naast hogere pagina dan die van het basisadres. Is het basisadres 02E5 en de inhoud van het betreffende indexregister \$ 20 dan is het aangewezen adres inderdaad

$$02E5 + 20 = 0305.$$

Liggen het basisadres en het uiteindelijke adres op dezelfde pagina dan zijn de handelingen door de processor als volgt (LDA op adres 0200, basisadres is 0310, $x = \$ 05$):

klok cyclus	adres bus	data bus handeling	externe handeling	interne
1.	0200	BD (LDA)	Fetch op-code	Voltooi voorgaande instructie. PC wordt 0201
2.	0201	10 (BAL)	Fetch BAL	Decodeer LDA PC wordt 0202
3.	0202	03 (BAH)	Fetch BAH	ADL wordt $BAL + x =$ $10 + 05 = 15$ PC wordt 0203
4.	0315	data	Fetch data	
5.	0203	volgende op-code	Fetch op-code	Voltooi LDA. PC wordt 0204

Komt het uiteindelijke adres een pagina hoger uit dan die van het basisadres dan is nog een extra klokcyclus nodig om na de optelling $BAL + x$ ook nog de carry bit (1) bij BAH op te tellen, zodat $ADH = BAH + 1$. Deze adressering kan bij toepassing van het index-x-register gebruikt worden bij:

ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC en STA.

En bij toepassing van het index-y-register:

ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC en STA.

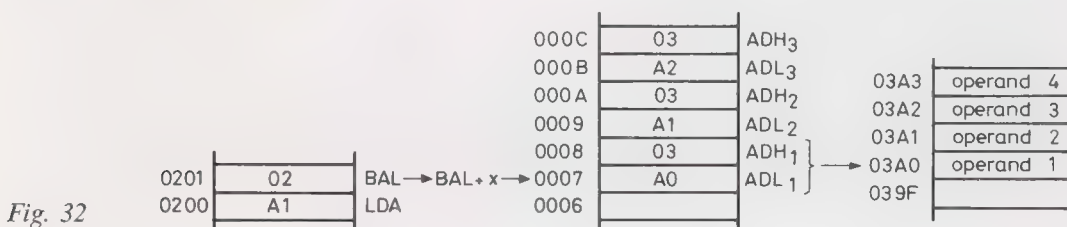
4.3. De (indirect, x) addressing

Zoals uit de titel blijkt betreft het hier een indirecte adressering met een indexing waarbij het index-x-register gebruikt wordt. De werkwijze is als volgt:

Na de op-code volgt een adres op pagina \$ 00 (zero page addressing). Dit is het basisadres. Hierbij wordt de inhoud van het indexregister opgeteld. Het indirecte adres is dan gevonden en daarmee het adres waarin de operand is geladen.

In fig. 32 vinden we op adres 0200 de op-code voor LDA met daaraanvolgend \$ 02 als basisadres laag (BAL). Hierbij wordt de inhoud van het index-x-register (hier 05) opgeteld, dus

$$02 + 05 = 07.$$



Dit is de lage byte van het indirecte adres op pagina \$ 00 (IAL_1).

Op dit adres staat de lage byte van het adres van operand 1 (ADL_1). De hoge byte (ADH_1) staat in geheugenplaats 0008. ADH_1 en ADL_1 vormen samen het adres (03A0) van operand 1.

Om het adres van de volgende operand te vinden (operand 2) moet de inhoud van het index-x-register met 2 worden verhoogd. In dit geval

$$05 + 02 = 07.$$

De lage byte van het indirecte adres is dan

$$02 + 07 = 09.$$

Op het indirecte adres 0009 vinden we ADL_2 en op 000A ADH_2 .

ADH_2 en ADL_2 vormen samen het adres van de operand 2.

Er zijn hier betrekkelijk veel geheugenplaatsen nodig. Bij tien verschillende operanden zijn twintig geheugenplaatsen nodig voor de indirecte adressen op pagina \$ 00. De operanden kunnen zich echter op elke willekeurige plaats in het geheugen bevinden. Voor deze adresseermethode die wel 'indexed indirect addressing' wordt genoemd, moet de processor de volgende handelingen verrichten ($x = \$ 05$):

klok cyclus	adres bus	data bus	externe handelingen	interne handelingen
1.	0200	A1 (LDA)	Fetch op-code	Voltooi vorige instructie. PC wordt 0201
2.	0201	02 (BAL)	Fetch BAL	Decodeer LDA PC wordt 0202
3.	0002 (00, BAL)	xxx	xxx	IAL_1 wordt $BAL + x_1$
4.	0007 (00, IAL_1)	A0 (ADL_1)	Fetch ADL_1	Verhoog IAL_1 met 1
5.	0008	03 (ADH_1)	Fetch ADH_1	Houdt ADL_1
6.	03A0	operand	Fetch operand 1	
7.	0202	Op-code	Fetch op-code	Voer instructie LDA uit. PC wordt 0203

De processor heeft een extra klokcyclus nodig (cyclus 3) voor het optellen van x bij BAL.

De instructies waarbij deze adresseermethode kan worden toegepast zijn:

ADC, AND, CMP, EOR, LDA, ORA, SBC en STA.

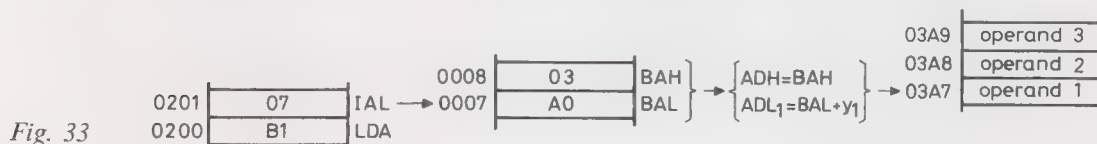
4.4. De (indirect), y addressing

Deze adresseermethode wijkt in zeker opzicht af van de (indirect, x) addressing en niet alleen omdat hierbij het index-y-register wordt gebruikt. Na de op-code volgt nu direct de laagstwaardige byte van het indirecte adres op pagina \$ 00 (dus ook zero page addressing). Hierin staat het basisadres laag (BAL). In de volgende geheugenplaats is het basisadres hoog (BAH) opgeslagen. Het gehele basisadres is daarom bekend en door bij BAL de inhoud van het index-y-register op te tellen wordt het operandadres gevonden ($ADH = BAH$, $ADL = BAL + y$).

In fig. 33 staat op adres 0200 de op-code van de LDA-instructie, gevolgd door 07 op 0201. Dit laatste is de lage byte van het indirecte adres op pagina \$ 00 (IAL). Op dit indirecte adres staat de lage byte van het basisadres (BAL). Het volgende adres heeft de hoge byte van het basisadres als inhoud (BAH op 0008). Door BAL te verhogen met de inhoud van het index-y-register (hier \$ 07) wordt het adres van de operand gevonden ($ADH_1 = BAH$, $ADL_1 = BAL + y_1$).

Om de tweede operand te vinden, moet het index-y-register met 1 worden verhoogd. Bij deze adresseermethode zijn slechts twee geheugenplaatsen op pagina \$ 00 nodig om alle operanden te kunnen adresseren. De operanden kunnen echter niet verder dan 256 geheugenplaatsen (max. inhoud van het index-y-register) uit elkaar liggen. Uitgaande van de gegevens, zoals bij fig. 33, verricht de processor de volgende handelingen:

klok cyclus	adres bus	data bus	externe handeling	interne handeling
1.	0200	B1 (LDA)	Fetch op-code	Voltooi voorgaande instructie. PC wordt 0201
2.	0201	07 (IAL)	Fetch IAL	Decodeer LDA. PC wordt 0202



klok cyclus	adres bus	data bus	externe handeling	interne handeling
3.	0007 (00, IAL)	A0 (BAL)	Fetch BAL	Verhoog IAL met 1
4.	0008	03 (BAH)	Fetch BAH	ADL wordt BAL+y
5.	03A7 (ADH, ADL)	data	Fetch operand	
6.	0202	op-code	Fetch op-code	Voer instructie LDA uit. PC wordt 0203

Er kunnen operanden in geheugenplaatsen van twee elkaar opvolgende pagina's worden opgeslagen. In dat geval geeft BAH een te laag paginanummer aan. Nu is een extra klokcyclus nodig om — na de berekening $ADL = BAL + y$ — de hoge byte van het adres te bepalen uit $ADH = BAH + 1$. De instructies waarop deze adressering kan worden toegepast, zijn:

ADC, AND, CMP, EOR, LDA, ORA, SBC en STA.

Lijst 2. Instructieset 6510 processor

INSTRUCTIONS		Immediate	Absolute	Zero Page	Accum.	Implied	(Ind.) X	(Ind.) Y	Z. Page, X	Abs X	Abs Y	Relative	Indirect	Z. Page, Y	CONDITION CODES					
Mnemonic	Operation	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	OP N	N	Z	C	I	D	V
ADC	A ← M ← C ← A (4)	69 2 2	6D 4 3	65 1 2			61 6 2	71 5 2	75 4 2	7D 4 3	79 4 3									
AND	A ← M ← A (1)	29 2 2	2D 4 3	25 3 2			21 6 2	31 5 2	35 4 2	3D 4 3	39 4 3									
ASL	C ← 0 ← 0		0E 6 3	06 5 2	0A 2 1				16 6 2	1E 7 3										
BCC	BRANCH ON C = 0 (2)											90 2 2								
BCS	BRANCH ON C = 1 (2)											B0 2 2								
BEO	BRANCH ON Z = 1 (2)											F0 2 2								
BIT	A ← M		2C 4 3	24 3 2																
BMI	BRANCH ON N = 1 (2)											30 2 2								
BNE	BRANCH ON Z = 0 (2)											D0 2 2								
BPL	BRANCH ON N = 0 (2)											10 2 2								
BRK						00 7 1														
BVC	BRANCH ON V = 0 (2)											50 2 2								
BVS	BRANCH ON V = 1 (2)											70 2 2								
CLC	0 ← C					18 2 1														
CLD	0 ← D					D8 2 1														
CLI	0 ← I					58 2 1														
CLV	0 ← V					B8 2 1														
CMP	A ← M (1)	C9 2 2	CD 4 3	C5 3 2			C1 6 2	D1 5 2	D5 4 2	DD 4 3	D9 4 3									
CPX	X ← M	E0 2 2	EC 4 3	E4 3 2																
CPY	Y ← M	C0 2 2	CC 4 3	C4 3 2																
DEC	M ← 1 ← M		CE 6 3	C6 5 2						D6 6 2	DE 7 3									
DEX	X ← 1 ← X					CA 2 1														
DEY	Y ← 1 ← Y					B8 2 1														
EOR	A ← M ← A (1)	49 2 2	4D 4 3	45 3 2			41 6 2	51 5 2	55 4 2	5D 4 3	59 4 3									
INC	M ← 1 ← M		EE 6 3	E6 5 2						F6 6 2	FE 7 3									
INX	X ← 1 ← X					E8 2 1														
INY	Y ← 1 ← Y					C8 2 1														
JMP	JUMP TO NEW LOC		4C 3 3										6C 5 3							
JSR	JUMP SUB		20 6 3																	
LDA	M ← A	A9 2 2	AD 4 3	A5 3 2			A1 6 2	B1 5 2	B5 4 2	BD 4 3	B9 4 3									
LDX	M ← X (1)	A2 2 2	AE 4 3	A6 3 2							BE 4 3			B6 4 2						
LDY	M ← Y	A0 2 2	AC 4 3	A4 3 2						B4 4 2	BC 4 3									
LSR	0 ← 7 ← 0 ← C		4E 6 3	46 5 2	4A 2 1					56 6 2	5E 7 3									
NOP	NO OPERATION					EA 2 1														
ORA	A ← M ← A (1)	09 2 2	0D 4 3	05 3 2			01 6 2	11 5 2	15 4 2	1D 4 3	19 4 3									
PHA	A ← M, S ← 1 ← S					48 3 1														
PHP	P ← M, S ← 1 ← S					08 3 1														
PLA	S ← 1 ← S, M ← A					68 4 1														
PLP	S ← 1 ← S, M ← P					28 4 1														
ROL			2E 6 3	26 5 2	2A 2 1					36 6 2	3E 7 3									
ROR			6E 6 3	66 5 2	6A 2 1					76 6 2	7E 7 3									
RTI	RTRN INT					40 6 1														
RTS	RTRN SUB					60 6 1														
SBC	A ← M ← A (1)	E9 2 2	ED 4 3	E5 3 2			E1 6 2	F1 5 2	F5 4 2	FD 4 3	F9 4 3									
SEC	1 ← C					38 2 1														
SED	1 ← D					F8 2 1														
SEI	1 ← I					78 2 1														
STA	A ← M		8D 4 3	85 3 2			81 6 2	91 5 2	95 4 2	9D 5 3	99 5 3									
STX	X ← M		8E 4 3	86 3 2																
STY	Y ← M		8C 4 3	84 3 2						94 4 2										
TAX	A ← X					AA 2 1														
TAY	A ← Y					AB 2 1														
TSX	S ← X					BA 2 1														
TXA	X ← A					8A 2 1														
TXS	X ← S					9A 2 1														
TYA	Y ← A					98 2 1														

(1) ADD 1 TO N IF PAGE BOUNDARY IS CROSSED
(2) ADD 1 TO N IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO N IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT BORROW
(4) IF IN DECIMAL MODE Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M₂ MEMORY PER STACK POINTER

+ ADD
- SUBTRACT
A AND
V OR
V EXCLUSIVE OR

✓ MODIFIED
- NOT MODIFIED
M₂ MEMORY BIT 7
M₆ MEMORY BIT 6
N NO CYCLES
NO BYTES

5. Inleiding

5.1. Het invoeren van machinetaalprogramma's

Zoals u weet heeft de Commodore 64 geen ingebouwde mogelijkheid om machinetaalprogramma's in te voeren. Nu kan het invoeren van een kort machinetaalprogrammaatje als onderdeel van een BASIC programma, nog wel met POKE-opdrachten geschieden. Voor langere programma's is dit totaal ondoenbaar. Hiervoor zult u gebruik moeten maken van hulpprogramma's zoals

64MON of SUPERMON 64, die u bij uw dealer kunt aanschaffen. Deze programma's hebben behalve de mogelijkheid van het invoeren, ook nog vele andere functies die bij het machinetaalprogrammeren erg gemakkelijk zijn, zoals assembleren en disassembleren. Bent u nog niet in het bezit van een dergelijk programma dan kunt u in elk geval vast beginnen in machinetaal te programmeren met behulp van het volgende BASIC programma.

```

5 DIM MT(47): FOR X=1 TO 47: READ MT(X): POKE 4096+X,MT(X): NEXT
10 PRINT CHR$(147): POKE 1417,0: DEF FNI(X)=X AND 1
20 INPUT "WAT IS HET STARTADRES": AD
25 PRINT CHR$(147)
30 PRINT AD: D=(PEEK(AD) AND 240)/16: HT=D: GOSUB 1000
60 D1=D: DA=PEEK(AD): D=DA AND 15: LT=D: GOSUB 1000
80 D0=D: PRINT CHR$(D1): CHR$(D0): X=1
90 GET D$: IF D$="" THEN PRINT CHR$(145): GOTO 30
100 E=ASC(D$): IF E=13 THEN 10
110 IF E=133 THEN 200
115 IF E=134 THEN X=0: GOTO 310
120 IF E=43 THEN 180
130 IF E=45 THEN 190
140 E=E-55: IF E<7 OR E>15 THEN 90
150 IF E>9 THEN 170
160 E=E+7: IF E>9 THEN 90
170 DH=(PEEK(AD) AND 15)*15+E: POKE AD,DH: PRINT CHR$(145): GOTO 30
180 AD=AD+1: GOTO 30
190 AD=AD-1: GOTO 30
200 BR=0: IF LT=12 AND HT=0 THEN 270
205 IF LT=0 AND HT=8 OR LT=9 AND HT=8 OR LT=14 AND HT=9 THEN 270
205 IF LT=8 AND (HT=15 OR HT<7 AND FNI(HT)=0) OR LT=10 AND HT=9 THEN 480
210 IF LT=0 AND FNI(HT)=1 THEN A=2: GOTO 280
220 IF LT=0 AND HT<8 OR LT=12 AND (HT=4 OR HT=6) THEN 470
230 IF LT=8 OR LT=10 AND (HT=9 OR HT=11 OR FNI(HT)=0) THEN A=1: GOTO 300
240 IF LT=0 AND HT<8 OR LT=1 OR LT=5 OR LT=6 OR LT=9 AND FNI(HT)=0 THEN 500
250 IF LT=2 AND HT=10 OR LT=4 AND (HT=2 OR HT=14 OR HT>7 AND HT<13) THEN 500
260 IF LT=13 OR LT=12 AND (FNI(HT)=0 OR HT=11) THEN 510
265 IF LT=9 AND FNI(HT)=1 OR LT=14 THEN 510
270 GOTO 460
280 BR=PEEK(AD+1): IF (BR AND 128)=128 THEN BR=BR-256
300 S3=PEEK(AD+A+BR): S4=PEEK(AD+A+BR+2)
305 S1=PEEK(AD+A): S2=PEEK(AD+A+2)
308 POKE AD+A+BR,0: POKE AD+A,0
310 AH=INT(AD/256): AL=AD-AH*256
320 POKE 4445,AH: POKE 4444,AL: POKE 790,69: POKE 791,17: SYS 4400
340 POKE 790,102: POKE 791,254: IF X=0 THEN 380
360 POKE AD+A+BR,S3: POKE AD+A+BR+2,S4
370 POKE AD+A,S1: POKE AD+A+2,S2
380 PRINT CHR$(147): D$(0)="A": D$(1)="X": D$(2)="Y": FOR X=0 TO 2
390 A=4448+X: D=(PEEK(A) AND 240)/16: GOSUB 1000: D1=D
400 D=PEEK(A) AND 15: GOSUB 1000: D0=D
410 PRINT SPC(20): D$(X): SPC(2): CHR$(D1): CHR$(D0): NEXT

```

```

420 PRINT SPC(23);CHR$(78);SPC(1);CHR$(86);SPC(3);CHR$(66);SPC(1);CHR$(69);
425 PRINT SPC(1);CHR$(73);SPC(1);CHR$(90);SPC(1);CHR$(67)
430 PRINT SPC(20);"P";SPC(1);
440 A=256:D=PEEK(4447):FOR X=0 TO 7:A=A/2:PRINT (D AND A);A;CHR$(157):NEXT
450 AD=PEEK(4445)*256+PEEK(4444)-2:PRINT CHR$(19):GOTO 30
460 PRINT CHR$(147):PRINT "DATA: GEEN OP-CODE":GOTO 490
470 PRINT CHR$(147):PRINT "SPRONGOPDRACHT, GEEF NIEUW ADRES"
475 FOR Y=0 TO 3000:NEXT :GOTO 10
480 PRINT CHR$(147):PRINT "OP-CODE NIET TOEGESTAAN BIJ DEBUGGEN"
490 FOR Y=0 TO 3000:NEXT :GOTO 25
500 A=2:GOTO 300
510 A=3:GOTO 300
1000 D=D+48:IF D<58 THEN 1300
1200 D=D+7
1300 RETURN
1400 DATA 186,8,173,95,17,157,0,1,172,98,17,174,97,17,173,96,17,40,100,32,17,100
1500 DATA 4,104,153,94,17,136,208,249,186,232,104,141,92,17,104,141,90,17
1600 DATA 154,169,96,141,0,0,64

```

De gebruiksaanwijzing van het programma is eenvoudig: na het invoeren van dit BASIC programma en RUN wordt u verzocht het startadres van het machinetaalprogramma in te voeren. Voldoet u hieraan (decimaal) dan verschijnt linksboven op het beeldscherm dit adres in decimale vorm met daarachter de inhoud hiervan als een hexadecimaal getal. Met de toetsen 0 tot en met 9 en A tot en met F kunt u deze inhoud veranderen en zodoende in deze geheugenplaats het hexadecimale getal invoeren van een operatiecode of een operand. Met de '+' toets kunt u het getoonde adres met 1 verhogen en met de '-' toets met 1 verlagen. Het nieuwe adres met inhoud verschijnt steeds onder het voorgaande zodat bij regelmatig verhogen of verlagen een rij adressen met inhoud worden afgebeeld waarvan de inhoud van het laatste adres steeds kan worden veranderd. Wilt u een geheel nieuw adres invoeren dan kan dat na de toets RETURN te hebben ingedrukt.

Een programma dat op deze manier is ingevoerd kan op de juiste werking worden gecontroleerd door het stap voor stap te laten uitvoeren. Dat wil zeggen dat niet het gehele programma in één keer wordt doorlopen maar dat steeds slechts één instructie van het programma wordt uitgevoerd. Door de functietoets f1 te bedienen wordt de operatiecode uitgevoerd die is opgeslagen in het laatste adres dat op het beeldscherm wordt getoond. Na het uitvoeren van deze ene operatie wordt rechts op het scherm de inhoud van de processorregisters A, X, Y en P getoond, de eerste drie hexadecimaal en de laatste binair, zodat de waarde van

elk bit afzonderlijk is af te lezen. De inhoud van deze registers kan gewijzigd worden op onderstaande adressen:

```

P 4447
A 4448
X 4449
Y 4450

```

(P is het conditie code register)

Verder wordt op het beeldscherm het adres en de inhoud van de eerst volgende operatiecode van het te testen programma getoond, dat bij het opnieuw indrukken van de toets f1 op zijn beurt zal worden uitgevoerd.

De constructie van dit programma geeft enkele beperkingen, zodat sommige, voor het testen minder belangrijke instructies niet kunnen worden uitgevoerd. Dit behoeft geen belemmering te zijn om dat het programma ook in gedeelten gecontroleerd kan worden. Wilt u een programma tot aan een bepaalde instructie in zijn geheel controleren dan dient u deze instructie te vervangen door de BRK-instructie (00). Bij het bedienen van de f3 toets wordt dan het programma doorlopen vanaf het ingevoerde startadres tot aan de BRK-instructie, waarna de inhoud van de processorregisters weer worden getoond. Hiervoor moet dan ook nog twee geheugenplaatsen na de BRK-instructie een RTS-instructie worden geplaatst (\$ 60). Belangrijk is dat in hetzelfde programmagedeelte eventuele PUSH-instructies worden gevolgd door de bijbehorende PULL-instructies.

5.2. Het save en loaden van machinetaalprogramma's

Het save en loaden van machinetaalprogramma's is met enige voorbereiding op gelijke wijze uit te voeren als het save en loaden van BASIC programma's. Deze 'voorbereidingen' behoeven enige toelichting. Het zal u bekend zijn dat de BASIC geheugenruimte begint op geheugenplaats 2048 (\$0800). Dit is overigens niet zo vanzelfsprekend. Dit aanvangsadres wordt de computer aangewezen door een vector die bij het 'opstarten' van de computer (na het inschakelen van de voedingsspanning) wordt geplaatst in de geheugenplaatsen 43 en 44 (TXTTAB, Start off BASIC text). Probeer maar eens in de direct mode:

```
PRINT PEEK(43); PEEK(44)
```

Het resultaat zal 1 en 8 zijn. De vectoren zijn steeds in de volgorde lage byte — hoge byte in de geheugenplaatsen opgeslagen. Het resultaat is dus $8 \cdot 256 + 1 = 2049$. Dit getal vormt het eerste adres dat wordt gebruikt voor het BASIC programma. Het adres 2048, dat wordt gegeven als het aanvangsadres van de BASIC geheugenruimte, is gevuld met het getal 0. Dit is een voorwaarde, het adres precies voór dat waarop het BASIC programma aanvangt, moet gevuld zijn met het getal 0. Probeer maar:

```
PRINT PEEK(2048)
```

Er is nog een andere vector te noemen, namelijk de vector die het eind van het ingevoerde BASIC programma aangeeft. Omdat de in het programma gebruikte variabelen steeds na het BASIC programma worden opgeslagen, is deze vector ook het adres voor het begin van de variabelen. Probeer het volgende eens:

```
NEW: PRINT PEEK(45); PEEK(46)
```

De vector in deze geheugenplaatsen (VARTAB, Start off BASIC Variables) is $3 + 8 \cdot 256 = 2051$. Door het invoeren van NEW is er geen programma meer in de computer aanwezig. Deze vector wordt bij het invoeren van een programma middels het toetsenbord of na een LOAD commando, steeds opgehoogd. Toets maar eens een willekeurig programma in en ga na het invoeren van elke programmaregel (dus na RETURN) maar eens na wat de waarde van deze vector is.

Het zijn de nu genoemde vectoren die door de SAVE routine worden overgenomen als het begin en het eindadres van het te save programma. Nu kunnen we de inhoud van deze vectoren veranderen en hierin het begin en het eindadres van het geheugenblok invoeren waarin een machinetaalprogramma is opgeslagen. Dit programma kan dan met het normale SAVE commando naar de tape of de floppy disk worden geschreven. Dit is te demonstreren met het volgende programma:

```
10 FOR X=0 TO 255
20 POKE 49152+X,X
30 NEXT X
```

Met dit programma vullen we de geheugenplaatsen 49152 tot en met 49407 met oplopende getallen. Deze vormen het te save geheugenblok. Het beginadres (BAD) moet nu gesplitst worden in een hoge byte (HBAD) en een lage byte (LBAD):

$$\text{HBAD} = \text{INT}(49152/256) = 192$$
$$\text{LBAD} = 49152 - 256 \cdot 192 = 0$$

Hiermee worden de geheugenplaatsen 44 en 43 gevuld. Nu het eindadres, het *eerste* adres *boven* het geheugenblok ($\text{EAD}, 49407 + 1 = 49408$):

$$\text{HEAD} = \text{INT}(49408/256) = 193$$
$$\text{LEAD} = 49408 - 256 \cdot 193 = 0$$

Voordat het geheugenblok wordt weggeschreven moeten we eerst de vectoren plaatsen:

```
POKE 43,0: POKE 44,192: POKE 45,0: POKE 46,193
```

U ziet, de vectoren worden zoals het hoort in de volgorde lage byte — hoge byte in de geheugenplaatsen geschreven.

Voor het schrijven naar de tape gebruikt u het volgende commando:

```
SAVE"VOORBEELD",1,1
```

Hier moeten een zogenaamde 'eerste adres' en 'tweede adres' aan het SAVE commando worden toegevoegd. Het eerste adres is voor de device nummer (1 voor het cassettedek) en het tweede (ook een 1) om later bij het lezen van de band (LOAD) het programma weer in de oorspronkelijke geheugenplaatsen terug te krijgen. Voor het

schrijven naar de disk heeft u het volgende commando nodig:

```
SAVE"VOORBEELD",8,1
```

Wilt u nu weer terug naar de oude situatie dan gebruikt u de volgende opdrachten:

```
POKE 43,1: POKE 44,8: NEW
```

Nu kan het zijn dat u een BASIC programma in de computer hebt dat u niet verloren wilt laten gaan. In dat geval gaat u, voordat u de vectoren van het te saven geheugenblok invoert, eerst na wat de inhoud van de geheugenplaatsen 45 en 46 is met

```
PRINT PEEK(45); PEEK(46)
```

Na het saven herstelt u de oude situatie nu met

```
POKE 43,1: POKE 44,8: POKE 45,P: POKE 46,Q
```

Waarin P en Q de oorspronkelijke inhoud van de geheugenplaatsen 45 en 46.

Voordat we het loaden van het geheugenblok van de band of van de disk gaan proberen schakelen we eerst de computer uit en weer aan, zodat de inhoud van het te vullen geheugenblok volkomen willekeurig is. Nu laden we het programma van de tape met:

```
LOAD"VOORBEELD",1,1
```

of van de disk met

```
LOAD"VOORBEELD",8,1
```

Lees nu de inhoud van de geheugenplaatsen 45 en 46 eens met

```
PRINT PEEK(45); PEEK(46)
```

U ziet dat deze het eindadres van het geheugenblok aangeven. Toets nu eens in:

```
10 FOR X=0 TO 255
```

Dit lukt niet, de computer laat het afweten. U dient eerst het commando NEW in te voeren. Nu kunt u het volgende programma proberen:

```
10 FOR X=0 TO 255  
20 PRINT PEEK(49152+X);  
30 NEXT X
```

U zult zien dat de geheugenplaatsen in het blok weer de juiste inhoud hebben gekregen.

5.3. Het gebruiken van de geheugenruimte

De gebruikelijke indeling van de geheugenruimte van de Commodore 64 geeft figuur 34. Het eerste blok is 40 Kbyte groot (\$ 0000 — \$ 9FFF) en wordt in beslag genomen door RAM geheugenelementen. Daarna volgt de BASIC interpreter in ROM (\$ A000 — \$ BFFF). Nu volgt een 4 Kbyte blokje dat weer gevuld is met RAM geheugenelementen (\$ C000 — \$ CFFF). Het volgende 4 Kbyte blok (\$ D000 — \$ DFFF) wordt gebruikt door de MOS 6566 VIC II, color RAM (geheugenplaatsen van vier bits) en de MOS 6526 Complex Interface Adapter (CIA) 1 en 2. In plaats hiervan kan door een commando ook de 4 Kbyte karakter ROM in dit blok worden geschakeld. Het laatste gedeelte (\$ E000 — \$ FFFF) wordt bezet door het systeemprogramma — de KERNAL — in ROM.

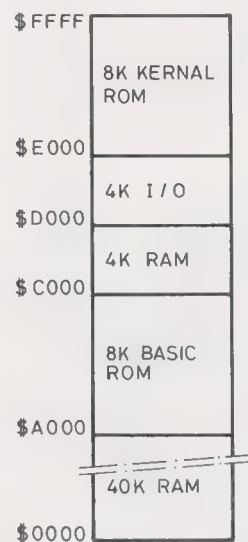


Fig. 34

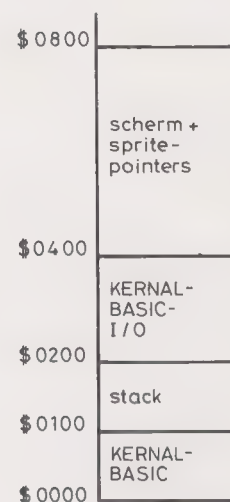


Fig. 35

Het 40 Kbyte RAM gedeelte is niet onbeperkt beschikbaar voor de gebruiker. Het gedeelte van \$ 0000 t.e.m. \$ 07FF wordt door de KERNAL, de VIC II-chip, de BASIC interpreter en de processor gebruikt. De indeling hiervan geeft figuur 35. Ten eerste pagina \$ 00 (\$ 0000 — \$ 00FF). Geheugenplaats \$ 0000 is de Data Direction Register

voor \$ 0001 en van \$ 0001 zijn 6 bits de uitgangspoorten van de 6510 processor. Op vier geheugenplaatsen na (\$ 00FB — \$ 00FE), die beschikbaar zijn voor de gebruiker, wordt de rest van de pagina gebruikt door de KERNAL en de BASIC interpreter.

Daarna volgt de STACK ruimte voor de processor. Hierboven zijn opnieuw twee pagina's (\$ 0200 — \$ 03FF) in gebruik bij de KERNAL en de BASIC interpreter. Vervolgens worden de volgende vier pagina's gebruikt voor het schermgeheugen en de Sprite Data Pointers. Vanaf \$ 0800 volgt de BASIC geheugenruimte.

De indeling zoals die in figuur 34 is gegeven kan worden gewijzigd.

Zowel de KERNAL als de BASIC interpreter kunnen worden in- en uitgeschakeld. Hiertoe dienen de poorten 0 en 1 van de processor 6510 die als uitgangspoorten zijn geschakeld (de bits 0 en 1 van DDR \$ 0000 zijn 1). Normaal zijn deze uitgangs-

poorten 1. Poort 0 (LORAM) schakelt de BASIC ROM in of uit. Is dit bit 0 dan is de BASIC ROM uitgeschakeld en een evengroot RAM blok in zijn plaats ingeschakeld.

Dit geldt voor elk ROM geheugenblok in de computer, dus ook voor de KERNAL. Wordt ook deze uitgeschakeld dan komt hiervoor eveneens een RAM geheugenblok in de plaats. Wordt naar een adres van een ROM geheugenblok geschreven dan wordt in werkelijkheid geschreven naar het RAM geheugen, dat op de achtergrond hiervan aanwezig is. Wordt het adres echter gelezen (bij ingeschakelde ROM) dan wordt de betreffende ROM geheugenplaats gelezen. Voer ter demonstratie het volgende commando eens in:

POKE 40960,X

Hierin kiest u voor X een bepaalde waarde ($40960 \triangleq \$ A000$). Nu voert u het volgende machinetaalprogramma in:

Het in- en uitschakelen van de BASIC interpreter

49408	C100	A9 26	LDA-IMM	26	# 38
49410	C102	85 01	STA-Z.PAGE		Uitschakelen BASIC interpreter.
49412	C104	AD 00A0	LDA-ABS	A000	Bepaal inhoud 9P. # 40960.
49415	C107	85 FB	STA-Z.PAGE		Plaats deze in 9P. # 251.
49417	C109	A9 27	LDA-IMM	27	# 39.
49419	C10B	85 01	STA-Z.PAGE		Schakel BASIC interpreter weer in.
49421	C10D	60	RTS-IMPL		RETURN.

Een machinetaalprogramma zal u in dit boek steeds op deze manier worden gegeven: Op elke regel vindt u eerst het adres in decimale vorm, dan dat adres in hexadecimale vorm. Nu volgt in hexadecimale getallen de operatie, gevormd uit de operatiecode met daar aan volgend eventueel één of twee bytes voor de operand.

Dit zijn dus de getallen die een deel van het programma vormen en die ingevoerd moeten worden als u niet over een assembler beschikt.

Nu volgt de operatie in assemblertaal. Bent u in het bezit van een assembler dan dient u deze uitdrukking in te voeren. Eventueel wordt hierna nog het adres gegeven van de operand waarop de operatie betrekking heeft. De regel wordt afgesloten met een kort commentaar.

Voer nu in de direct mode in:

SYS 49408: PRINT PEEK(251)

U zult zien dat u de waarde die in getal X hebt ingevoerd nu op uw scherm wordt geprint.

De werking is als volgt: Met POKE 40960,X hebt u de RAM geheugenplaats \$ A000 met het getal X ingeschreven, ondanks dat de BASIC ROM die op dit adres aanvangt, nog ingeschakeld is. Nu springt u met SYS 49408 naar de subroutine die op \$ C100 aanvangt. Daarin wordt eerst de BASIC interpreter uitgeschakeld (regels 49408 en 49410) door in geheugenplaats 1 een *even* getal te schrijven ($\$ 26 \triangleq 38$, bit 0 = 0). Nu wordt in regel 49412 geheugenplaats 40960 gelezen en de waarde daarvan in regel 49415 in geheugenplaats \$ 00FB geschreven. Daarna wordt de BASIC interpreter weer ingeschakeld. Door RTS komen we weer terug bij de ingevoerde BASIC regel waarin door PRINT PEEK(251) de inhoud van geheugenplaats \$ 00FB op het scherm wordt getoond.

De machinetaalprogramma's in dit boek zijn onder andere bestemd om de diverse mogelijkheden van uw computer te demonstreren. Ook demonstreren ze de werking van de instructies en de adresseermethoden.

De bedoeling is dat u ze start vanuit BASIC met SYS49408. Daarom zijn ze steeds afgesloten met RTS zodat altijd weer naar BASIC wordt teruggekeerd. Start u ze vanuit een assemblerprogramma dan dient u daarbij de gebruiksaanwijzing hiervan in acht te nemen. Vaak moeten de machinetaalprogramma's dan worden afgesloten met BRK (\$ 00). Ook het invoeren van de instructies in de assemblertaal dient volgens de gebruiksaanwijzing van de assembler plaats te hebben.

De KERNAL ROM kunt u op overeenkomstige wijze als de BASIC ROM uitschakelen. In dat geval dient u bit 1 van geheugenplaats 1 nul te maken, bijvoorbeeld door het getal 37 (\$ 25) naar deze geheugenplaats te schrijven. Behalve de

KERNAL is nu echter ook de BASIC ROM uitgeschakeld zodat verder alleen nog maar de in- en uitgangsl logica ter beschikking is. Op de plaats van de BASIC en de KERNAL ROM is nu RAM geheugen ingeschakeld.

Nu lijkt het er op dat u wel de beschikking kunt krijgen over een grote RAM geheugenruimte maar daarvoor dan de subroutines in de BASIC of de KERNAL ROM moet missen. Dit hoeft niet het geval te zijn. Als u bijvoorbeeld de BASIC ROM wilt blijven gebruiken dan laadt u in de RAM achter deze ROM slechts subroutines. Voordat één van deze subroutines wordt aangeroepen schakelt u eerst de BASIC ROM uit en na het doorlopen van de subroutines weer in, overeenkomstig het volgende voorbeeld:

Een subroutine achter de BASIC interpreter

40960 A000	A9 7A	LDA-IMM	7A	# 122
40962 A002	85 FB	STA-Z.PAGE		# 122 NAAR GP. # 251
40964 A004	60	RTS-IMPL		RETURN

Dit is dan een subroutine die wordt geladen in de RAM achter de BASIC ROM. Hij heeft tot doel om het getal \$ 7A (122) in geheugenplaats \$ 00FB te schrijven. Heeft u niet de mogelijkheid om dit programma in assemblertaal of in machinecode achter de BASIC ROM in te schrijven, gebruik

dan het volgende BASIC programma:

```
10 FOR X=0 TO 4: READ A
20 POKE 40960+X,A: NEXT
50 DATA 169,122,133,251,96
```

Voer nu ook het volgende programma in:

Gebruik van de subroutine achter de BASIC interpreter

49408 C100	A9 26	LDA-IMM	26	# 38
49410 C102	85 01	STA-Z.PAGE		Uitschakelen BASIC interpreter.
49412 C104	20 00A0	JSR-ABS	A000	Voor het laden van GP. #251 met 122.
49415 C107	A9 27	LDA-IMM	27	# 39.
49417 C109	85 01	STA-Z.PAGE		Schakel BASIC interpreter weer in.
49419 C10B	60	RTS-IMPL		RETURN.

Dit programma schakelt eerst de BASIC ROM uit, roept de subroutine op \$ A000 aan en schakelt uiteindelijk de BASIC ROM weer in. Met de volgende opdracht ziet u het resultaat:

SYS 49408: PRINT PEEK(251)

Uiteraard heeft u ook de mogelijkheid beide poorten (LORAM en HIRAM) van geheugenplaats \$ 0001 nul te maken. Hiervoor kunt u bijvoorbeeld het getal \$ 24 naar geheugenplaats 1 te schrijven. Hebt u hierbij niets op de Cartridge/expansion stekker aangesloten dan zijn zowel de BASIC ROM, de KERNAL ROM, de karakter ROM als de in- en uitgangsl logica afgeschakeld. U heeft nu 64

Kbyte RAM ter beschikking (met uitzondering van geheugenplaatsen 0 en 1). Indien u in- of uitvoer nodig heeft dan dient u poort 1 weer 1 te maken. Dit kan uiteraard nu alleen maar in een machinetaalprogramma plaatshebben.

Ook de indeling zoals die in figuur 35 is gegeven is te veranderen. Zo kunt u zelf bepalen welk gedeelte van de geheugenruimte voor uw BASIC programma wordt gereserveerd (als u dat in combinatie met een machinetaalprogramma gebruikt). Waar u het beginadres hiervan kunt laden is u al bekend, de geheugenplaatsen 43 (\$ 002B) voor LBAD en 44 (\$ 002C) voor HBAD:

POKE BAD-1,0: POKE 43,LBAD:
POKE 44,HBAD: NEW.

Ook het eindadres (EAD) van de BASIC geheugenruimte kunt u zelf vastleggen. Hiervoor zijn de geheugenplaatsen 55 (\$ 0037) voor LEAD en 56 (\$ 0038) voor HEAD:

POKE 55,LEAD: POKE 56,HEAD: CLR

Het commando CLR is nodig om de pointers voor de variabelen de juiste inhoud te geven.

De plaats van het schermgeheugen kunt u in principe ook zelf bepalen. Dit heeft enige toelichting nodig. Voor het adresseren van 65536 geheugenplaatsen zijn 16 adreslijnen nodig ($2^{16} = 65536$). De VIC II-chip krijgt echter slechts 14 adreslijnen en daarmee kunnen niet meer dan 16384 geheugenplaatsen worden geadresseerd (16 Kbyte). De 64 Kbyte geheugenruimte is daarom in vier secties verdeeld en uit elk van de vier secties kan de VIC II-chip zijn gegevens halen. Deze gegevens betreffen de schermcode uit het schermgeheugen en het karakterpatroon uit de karakter ROM of uit de karakter RAM voor de zelfgemaakte karakters. Uit welke sectie de gegevens worden gehaald is afhankelijk van een getal van twee bits (van 0 tot en met 3 decimaal) dat hier met A wordt aangegeven. In

lijst 3 vindt u de *aanvangsadressen* van de secties voor een bepaalde waarde van A.

Lijst 3. Aanvangsadres geheugensectie

A	Sectie(s)	Aanvangs- adres hex.	Aanvangs- adres dec.
3	0	\$ 0000	0
2	1	\$ 4000	16384
1	2	\$ 8000	32768
0	3	\$ C000	49152

De selectie van de sectoren komt tot stand via de poorten PA0 en PA1 van de Complex Interfase Adapter Chip 2 (CIA 2) die zich op het adres 56576 bevinden (\$ DD00). Eerst deze poorten tot uitgang maken:

POKE 56578,PEEK(56578) OR 3

Het Data Direction Register van de poorten op het adres 56576 is register 56578, waarvan de bits 0 en 1 de waarde 1 moeten hebben.

Nu de betreffende sectie selecteren:

POKE 56576,PEEK(56576) AND 252 OR A

In machinetaal is hiervoor het volgende programma nodig:

Keuze van de geheugensectie

49408 C100 A9 03 LDA-IMM	03 Bits 0 en 1 zijn 1.
49410 C102 0D 02DD ORA-ABS	DD02 Laagstwaardige bits: 11.
49413 C105 8D 02DD STA-ABS	DD02 PA0 en PA1 worden uitgangspoorten.
49416 C108 AD 00DD LDA-ABS	DD00 Inhoud register in accu.
49419 C10B 29 FC AND-IMM	FC Maskeer de bits 0 en 1.
49421 C10D 09 03 ORA-IMM	03 Getal A is # 3 voor sectie 0.
49423 C10F 8D 00DD STA-ABS	DD00 PA0 en PA1 selecteren sectie 0.
49426 C112 60 RTS-IMPL	RETURN.

De eerste drie regels van dit programma zorgen dat PA0 en PA1 uitgangspoorten worden. Daartoe wordt eerst het getal 3 in de accu geladen (binair 11). Door ORA in regel 49410 krijgen de bits 0 en 1 van het getal uit registers \$ DD02 de waarde 1. De bewerking vindt in de accu plaats zodat na STA in regel 49413 pas het juiste getal in register \$ DD02 wordt geschreven. Het volgende programmadeel zorgt voor de selectie van de sectie. Eerst wordt de inhoud van register \$ DD00 in de accu geladen. Daarna wordt door de ORA bewerking met het getal \$ FC de bits 0 en 1

de waarde 0 gegeven. Dan wordt het getal A ingevoerd. In dit programma heeft A de waarde 3. Het getal voor A kan echter ook 0, 1 of 2 zijn. Door STA in regel 49423 krijgen PA0 en PA1 de juiste waarde.

Dit is overigens een programma dat zich uitstekend laat volgen bij het stap voor stap uitvoeren door het BASIC programma dat aan het begin van dit hoofdstuk is gegeven.

Nu we weten hoe we de sectie kunnen kiezen (de computer kiest zelf bij het inschakelen steeds sectie

0) moeten we ook nog nagaan op welke plaats in de sectie zich het schermgeheugen en de karaktergenerator (karakter ROM of karakter RAM) zullen bevinden.

De werkwijze die de VIC II-chip volgt bij het printen van een karakter op het scherm is als volgt: De VIC II-chip haalt de schermcode (SC) op van de eerste plaats in het schermgeheugen, *berekent* waar hij het bitpatroon in de karaktergenerator kan vinden die bij deze schermcode hoort en plaatst het gevonden bitpatroon op de eerste kolom van de eerste regel van het scherm. Dit herhaalt zich voor elke volgende geheugenplaats in het schermgeheugen totdat het gehele scherm gevuld is, de spaties inbegrepen. Hierbij worden de gegevens voor de kleur waarin geprint wordt, gehaald uit de color RAM en uit de andere kleurgeheugens. Deze bevinden zich steeds in de geheugenruimte voor de in- en de uitgangslotica, van \$ D000 tot en met \$ DFFF. Anders is het met de plaats van het schermgeheugen en de karaktergenerator. Deze bevinden zich op een plaats in de ingeschakelde sectie, waarbij het beginadres van het schermgeheugen en de karaktergenerator worden aangegeven door bepaalde bits van geheugenplaats 53272 (\$ D018).

Eerst het schermgeheugen. Het scherm kan $40 \times 25 = 1000$ karakters bevatten. Er wordt echter steeds gewerkt met getallen die een macht van twee zijn ($2^0, 2^1, 2^2, 2^3$ enz.) zodat voor het schermgeheugen een blok van 1024 bytes ($2^{10} \triangleq 1$ Kbyte) wordt gereserveerd. Hiervan passen er 16 in een sectie (16 Kbyte) zodat een vier bits getal groot genoeg is om de plaats in de sectie aan te wijzen ($2^4 = 16$).

Stellen we het sectienummer op S (S = 3-A, zie lijst 3) dan vinden we het startadres van het schermgeheugen bij verschillende waarde van het geal B, dat de plaats van het schermgeheugen in de sectie aangeeft, in lijst 4.

De computer zal zelf steeds voor het aanvangsadres van het schermgeheugen, geheugenplaats 1024 (\$ 0400) in sectie 0 kiezen.

Wilt u het startadres zelf bepalen dan kunt u na het kiezen van de sectie het getal B invoeren in de vier hoogstwaardige bits van geheugenplaats 53272 (\$ D018) met

POKE 53272, PEEK(53272) AND 15 OR 16 * B

Lijst 4. Aanvangsadres schermgeheugen

B	Aanvangsadres (hex) in sectie 0	Aanvangsadres (dec) + S*16384
0	\$ 0000	0
1	\$ 0400	1024
2	\$ 0800	2048
3	\$ 0C00	3072
4	\$ 1000	4096
5	\$ 1400	5120
6	\$ 1800	6144
7	\$ 1C00	7168
8	\$ 2000	8192
9	\$ 2400	9216
10	\$ 2800	10240
11	\$ 2C00	11264
12	\$ 3000	12288
13	\$ 3400	13312
14	\$ 3800	14336
15	\$ 3C00	15360

U kunt ook meerdere blokken reserveren als schermgeheugen. De computer zal echter steeds één van deze blokken als schermgeheugen beschouwen (uitlezen). Door om te schakelen (veranderen van getal B in geheugenplaats 53272) wisselt u dan van schermgeheugen en daarmee de inhoud van het scherm.

Voor elk karakter op het scherm is slechts één byte in het schermgeheugen gereserveerd en daarom kan de schermcode van een karakter (SC) slechts een getal zijn van 0 tot en met 255. Dat betekent dat een karakterset kan bestaan uit 256 verschillende karakters. Voor het bitpatroon van één karakter zijn 8 bytes nodig zodat de karaktergenerator een blok van $256 * 8 = 2048$ bytes omvat. Er gaan daarvan in één sectie $16384/2048 = 8$ blokken. Er is daarom een getal van 0 tot en met 7 nodig voor het aangeven van de plaats van een karaktergenerator in een sectie. Lijst 5 geeft de aanvangsadressen.

De computer zal steeds het aanvangsadres voor de karaktergenerator kiezen op geheugenplaats 4096 in sectie 0. In dat geval wordt de karakter ROM ingeschakeld.

U kunt zelf het aanvangsadres bepalen door het getal voor C in lijst 5 in te voeren in de bits 1 tot en met 3 van geheugenplaats 53272. Het bit 0 van

deze geheugenplaats wordt niet gebruikt. De volgende opdracht is hiervoor nodig :

POKE 53272, PEEK(53272) AND 240 OR 2 * C

Lijst 5. Aanvangsadres karaktergenerator

C	Aanvangsadres (hex) in sectie 0	Aanvangsadres (dec) + S*16384
0	\$ 0000	0
1	\$ 0800	2048
2	\$ 1000	4096
3	\$ 1800	6144
4	\$ 2000	8192
5	\$ 2800	10240
6	\$ 3000	12288
7	\$ 3800	14336

De eerste geheugenplaats van het bitpatroon van een karakter uit de karaktergenerator wordt nu berekend met

$$BP = S*16384 + C*2048 + SC*8$$

Hierin is BP het aanvangsadres van het bitpatroon, S het sectienummer (S = 3-A), C het nummer van de karaktergenerator in de sectie en SC de schermcode.

In de meeste gevallen moet u de karakterset vullen met bitpatronen van zelfgemaakte karakters (of de bitpatronen overnemen uit de karakter ROM). Dit is niet het geval voor C=2 en C=3 in *sectie 0* en in *sectie 2*. In deze gevallen wordt de karakter ROM ingeschakeld in de plaats van de in- en de uitgangsl logica in het blok van \$ D000 tot en met \$ DFFF. Het aanvangsadres van set 1 is \$ D000, in binaire vorm: 1101 0000 0000 0000. De VIC II-chip kent echter slecht 14 adreslijnen en ontvangt dus het adres 01 0000 0000 0000 \triangleq \$ 1000.

Het startadres van set 1 van de karakter ROM bevindt zich dus schijnbaar op adres \$ 1000 (4096 dec) en de VIC II-chip haalt dus ook schijnbaar daar de bitpatronen vandaan (sectie 0). De set 2 in de karakter ROM heeft als startadres \$ D800 en rekent in sectie 0 met het schijnbare startadres \$ 1800 (6144 dec). Voor de sectie 2 geldt hetzelfde. In de secties 1 en 3 dient u bij C=2 en C=3 zelf de bitpatronen in te voeren.

Veronderstel dat we bij het invoeren van de getallen voor A, B en C deze hebben opgeslagen in respectievelijk de geheugenplaatsen \$ 00FB, \$ 00FC en \$ 00FD, dan worden deze getallen met het volgende machinetaalprogramma ingevoerd:

Selectie van sectie, schermgeheugen en karaktergenerator

49408 C100	A9 03	LDA-IMM	03	Bits 0 en 1 zijn 1.
49410 C102	0D 02DD	ORA-ABS	DD02	Laagstwaardige bits: 11.
49413 C105	8D 02DD	STA-ABS	DD02	PA0 en PA1 worden uitgangspoorten.
49416 C108	AD 00DD	LDA-ABS	DD00	Inhoud register in accu.
49419 C10B	29 FC	AND-IMM	FC	Maskeer de bits 0 en 1.
49421 C10D	05 FB	ORA-Z.PAGE		Getal A in bits 0 en 1 van de accu.
49423 C10F	8D 00DD	STA-ABS	DD00	PA0 en PA1 selecteren sectie 0.
49426 C112	A5 FC	LDA-Z.PAGE		Getal B in accu.
49428 C114	0A	ASL-ACCU		2*B in accu.
49429 C115	0A	ASL-ACCU		4*B in accu.
49430 C116	0A	ASL-ACCU		8*B in accu.
49431 C117	65 FD	ADC-Z.PAGE		8*B + C in accu.
49433 C119	0A	ASL-ACCU		16*B + 2*C in accu.
49434 C11A	8D 18D0	STA-ABS	D018	Selecteren schermgeh. en karakt. gen.
49437 C11D	60	RTS-IMPL		RETURN.

De eerste zeven regels zijn praktisch gelijk aan die van het vorige programma. In dit geval wordt echter het getal A voor de selectie van de sectie opgehaald uit geheugenplaats 251 (\$ 00FB, regel 49421).

In regel 49423 wordt getal B in de accu geladen. Daarna wordt getal B met 8 vermenigvuldigd door drie keer ASL-ACCU toe te passen. Omdat B

steeds kleiner is dan 16 zal na drie maal ASL het carrybit nul zijn zodat zonder CLC het getal C kan worden opgeteld bij 8 * B (regel 49431). Door nu nogmaals alle bits in de accu naar links te schuiven vinden we getal B in de vier hoogstwaardige bits(hoge tetrade) en getal C in de bits 1, 2 en 3. Uiteraard kunt u het schermgeheugen en de karaktergenerator slechts op die plaatsen kiezen die niet

al voor andere doeleinden worden gebruikt. Rekening moet bijvoorbeeld worden gehouden met de processorregisters \$ 0000 en \$ 0001, de processor stack van \$ 0100 tot en met \$ 01FF en de in- outlogica in het blok \$ D000 — \$ DFFF.

Vaak is veranderen van de normale situatie niet nodig, bijvoorbeeld als een machinetaalprogramma niet groter is dan 4 Kbyte. In dat geval is het dan onder te brengen in het blok \$ C000 — \$ CFFF.

Heeft u daarbij twee schermgeheugens nodig en behalve de normale karaktergenerator in de karakter ROM ook nog een karaktergenerator met eigengemaakte karakters dan kan de indeling zijn zoals in figuur 36.

Omdat het geheel zich afspeelt in sectie 0 behoeven we getal A niet in te voeren. Van schermgeheugen 1 naar schermgeheugen 2 kan worden overgeschakeld door het getal B in geheugenplaats 53272 van 1 (normaal) in 2 te veranderen (en eventueel weer terug). Door getal C in dezelfde geheu-

genplaats van 2 (set 1, normaal) of 3 (set 2) te veranderen in 4 wordt de eigengemaakte karakterset in het blok \$ 2000 — \$ 27FF gekozen. Hierboven begint de BASIC geheugenruimte, te markeren met

POKE 10240,0: POKE 43,1: POKE 44,40: NEW

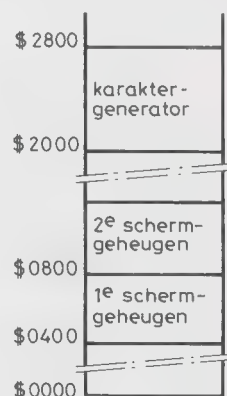


Fig. 36

6. Het in- en uitvoeren van variabelen

6.1. Inleiding

Met het invoeren van variabelen wordt hier het inbrengen van de gegevens via het toetsenbord bedoeld, gegevens die de computer nodig heeft voor het uitvoeren van een programma. Dit kunnen getallen zijn waarmee geheugenplaatsen moeten worden gevuld, integer en floating point variabelen, maar ook string variabelen. Omdat we nu eenmaal de subroutines van de systeemprogramma's in de commodore gebruiken zal ons de vergelijking met bepaalde BASIC commando's en BASIC statements worden opgedrongen. Bij het invoeren van de gegevens zijn dat de opdrachten GET A\$, VAL (A\$), INPUT A\$, INPUT A, READ A\$, READ A en POKE X, Y.

Met het uitvoeren van variabelen wordt hier het printen van een variabele (als uitkomst van een bewerking) op het scherm bedoeld. De belangrijkste vergelijkbare BASIC opdrachten hierbij zijn: PRINT A, PRINT A\$ EN PRINT PEEK(X).

Om in de programma's zoveel mogelijk een eenheid te brengen zullen in dit boek een aantal vaste geheugenplaatsen als beginadressen worden aangehouden.

De lengte van de geboden programma's zal beperkt zijn omdat het voorbeelden zijn op welke manier bepaalde subroutines kunnen worden gebruikt en hoe bepaalde situaties kunnen worden opgelost. Daarom hebben we steeds genoeg ruimte in het blok van \$ C000 tot en met \$ CFFF. Het aanvangsadres voor de programma's is \$ C100. Bij het invoeren van de karakters van het toetsenbord is een buffer nodig om die gegevens tijdelijk in op te slaan. Het aanvangsadres van deze buffer is \$ C800. Ook de (floating point) variabelen hebben hun vaste plaats in de geheugenruimte. Het beginadres van de eerste variabele is \$ C900 en voor elke variabele worden vijf geheugenplaatsen gereserveerd. Bij stringvariabelen is een tabel nodig. Deze laten we aanvangen op \$ CA00. Zoals gezegd, deze indeling zal door het gehele boek worden aangehouden.

6.2. Het in- en uitvoeren van karakters

Om gegevens via het toetsenbord te kunnen invoeren moet het systeemprogramma steeds naar het

toetsenbord 'kijken' of er soms een toets is ingedrukt, en welke toets dan is ingedrukt. Dit gebeurt niet continu maar elke 1/60 sec. Elke 1/60 sec. wordt door een 'interrupt' het dan lopende programma onderbroken en wordt overgeschakeld naar een programma, dat ook wel het 'interrupt programma' wordt genoemd, en dat het toetsenbord uitleest. Wordt op dat moment een toets ingedrukt dan wordt door het interruptprogramma de ASCII waarde van het karakter dat bij de toets hoort in het toetsenbord-buffergeheugen geplaatst. Deze buffer vinden we op de geheugenplaatsen \$ 0277 tot en met \$ 0280. In plaats van de ASCII waarde van een karakter zullen we verder eenvoudigweg spreken van 'een karakter'. Ook als een volgende toets wordt ingedrukt komt het karakter daarvan in het toetsenbordbuffergeheugen. In totaal kunnen hierin tien karakters worden opgeslagen. Uit het toetsenbordbuffergeheugen kan steeds één karakter worden gelezen met de subroutine GETIN. Door een aantal keren deze subroutine aan te roepen worden de karakters in de buffer één voor één gelezen. Het karakter dat het eerst in de buffer is geplaatst wordt het eerst gelezen. Steeds als een karakter wordt gelezen wordt een plaats vrij gemaakt voor een volgend karakter. Een routine die de werking heeft van GET A\$ en wacht op het indrukken van een toets

(10 GET A\$: IF A\$ = " " THEN 10) geeft het volgende programma. Figuur 37 geeft het stroomdiagram.

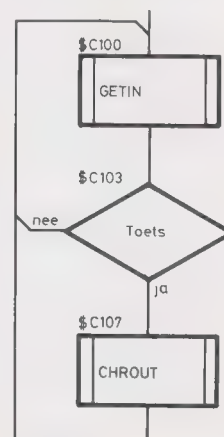


Fig. 37

GET karakter; PRINT karakter

49408	C100	20	E4FF	JSR-ABS	FFE4	GETIN routine, GET CHARACTER.
49411	C103	C9	00	CMP-IMM	00	Geen toets ingedrukt, \$ 00 in ACCU.
49413	C105	F0	F9	BEQ-REL	C100	Wacht op een toets.
49415	C107	20	D2FF	JSR-ABS	FFD2	PRINT het karakter.
49418	C10A	4C	00C1	JMP-ABS	C100	Wacht op de volgende toets.

Op regel 49408 wordt de subroutine GETIN aangeroepen. Deze haalt één karakter uit het toetsenbordbuffergeheugen. Dit karakter vinden we in de accu. Is de buffer leeg dan is de inhoud van de accu \$ 00 en wordt opnieuw naar regel 49408 gesprongen. Dit heeft het effect van het wachten op het indrukken van een toets. Is een toets ingedrukt dan wordt naar het volgende programmadeel overgegaan. Dit zorgt er voor dat u het ingevoerde karakter te zien krijgt. Daartoe moet het worden 'uitgevoerd' naar het scherm. Dit verzorgt de

subroutine CHROUT die in regel 49415 wordt aangeroepen. Deze verzorgt de uitvoer van het karakter in de accu naar het scherm en PRINT dus het karakter (steeds één per keer) op het scherm. U zult zien dat u, als u dit programma met SYS 49408 hebt aangeroepen, elk willekeurig karakter dat u intoetst, op het scherm te zien krijgt.

Deze routine kunnen we niet zo eenvoudig meer verlaten (alleen met de RUN/STOP-RESTORE toetsen). Dat gaat beter met het volgende programma:

Het verlaten van een programma

49408	C100	20	E4FF	JSR-ABS	FFE4	GETIN routine.
49411	C103	C9	00	CMP-IMM	00	Geen toets ingedrukt, \$ 00 in ACCU.
49413	C105	F0	F9	BEQ-REL	C100	Wacht op een toets.
49415	C107	C9	85	CMP-IMM	85	Toets f1 ingedrukt?
49417	C109	F0	06	BEQ-REL	C111	Zo ja, naar RETURN.
49419	C10B	20	D2FF	JSR-ABS	FFD2	PRINT het karakter.
49422	C10E	4C	00C1	JMP-ABS	C100	Wacht op de volgende toets.
49425	C111	60		RTS-IMPL		RETURN.

Het stroomdiagram hiervoor geeft figuur 38. Steeds als een toets is ingedrukt wordt de waarde van het getal in de accu vergeleken met \$ 85 (CHR\$(133) voor de f1 toets). De waarde van de accu komt hiermee overeen als de f1 toets is ingedrukt, zodat dan het programma wordt verlaten. Nu hebben we aan het hele programma niets als

we de ingetoetste karakters niet ergens in het geheugen opslaan. Het hierna volgende programma schrijft de ingetoetste karakters in de geheugenplaatsen die voor een tabel zijn gereserveerd. De karakters vormen samen namelijk een string. Het stroomdiagram geeft figuur 39.

GET karakterstring

49408	C100	A2	00	LDX-IMM	00	Voor de lengte van de string.
49410	C102	8A		TXA-IMPL		Stringlengte in ACCU.
49411	C103	48		PHA-IMPL		Save de stringlengte.
49412	C104	20	E4FF	JSR-ABS	FFE4	GETIN routine.
49415	C107	C9	00	CMP-IMM	00	Geen toets ingedrukt, \$ 00 in de ACCU.
49417	C109	F0	F9	BEQ-REL	C104	Wacht op een toets.
49419	C10B	C9	85	CMP-IMM	85	Toets f1 ingedrukt?
49421	C10D	F0	0D	BEQ-REL	C11C	Zo ja, naar RETURN.
49423	C10F	20	D2FF	JSR-ABS	FFD2	Print het karakter.
49426	C112	A8		TAY-IMPL		Bewaar het karakter in register Y.
49427	C113	68		PLA-IMPL		Haal stringlengte weer op.
49428	C114	AA		TAX-IMPL		Stringlengte in X-register.
49429	C115	98		TYA-IMPL		Karakter in de ACCU.
49430	C116	9D	00CA	STA-ABS,X	CA00	Karakter in de tabel.
49433	C119	E8		INX-IMPL		Verhoog de stringlengte.
49434	C11A	D0	E6	BNE-REL	C102	Voor volgend karakter.
49436	C11C	68		PLA-IMPL		Corrigeer de StackPointer.
49437	C11D	60		RTS-IMPL		RETURN.

Omdat elk volgend karakter in een hogere geheugenplaats in de tabel moet worden opgeslagen moet de stringlengte in het X-register worden opgeslagen (regel 49408). Dit register wordt echter

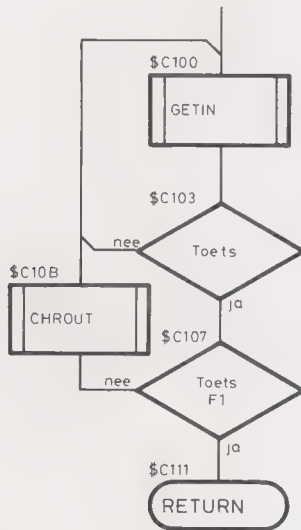


Fig. 38

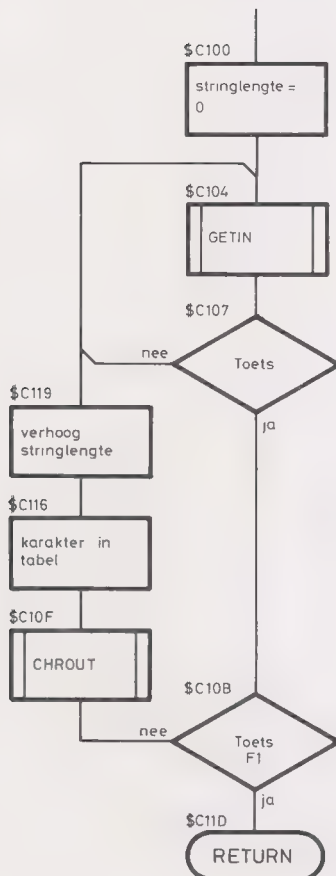


Fig. 39

ook in de subroutines gebruikt die verderop in het programma worden aangeroepen, zodat de stringlengte in de stack moet worden bewaard (regel 49411). De regels 40412 tot en met 49423 zijn al bekend. Hierna wordt de stringlengte weer uit de stack opgehaald zodat in regel 49430 het karakter op de plaats X in de tabel kan worden geschreven. Hierna wordt de stringlengte verhoogd voor het volgende karakter. Het X-register kan daardoor niet nul zijn zodat BNE het programma laat herhalen. Wordt het programma met de f1 toets verlaten dan moet eerst met PLA de stackpointer worden gecorrigeerd (op PHA in regel 49411 moet steeds een PLA volgen).

Hier wordt het programma verlaten met de f1 toets. Dat kan ook met een andere toets, bijvoorbeeld de RETURN toets. In dit geval is dat eigenlijk veel vanzelfsprekender. Dan moet echter het getal \$ 85 in regel 49419 worden veranderd in \$ 0D.

6.3. Het werken met het toetsenbord

De toetsen van het toetsenbord zijn in principe niet anders dan schakelaars die worden gesloten als de bijbehorende toetsen worden ingedrukt. Om te kunnen 'uitlezen' welke toets is ingedrukt zijn ze in een 8×8 rooster, een zogenaamde 'matrix' geplaatst (figuur 40). Dit is een rooster van acht verticale geleiders (de kolommen) die op de *uitgangspoorten* van register 56320 (\$ DC00) zijn aangesloten. Dit is een register van de Complex Interface Adapter (1). Acht horizontale geleiders (die niet met de verticale zijn verbonden) zijn aangesloten op de *ingangspoorten* van het register 56321 (\$ CD01) van hetzelfde IC CIA (1) en vormen de regels. Op de kruisingen bevinden zich de schakelaars van de toetsen. In figuur 40 zijn hiervan enkele getekend. Maximaal kunnen daardoor $8 \times 8 = 64$ schakelaars worden toegepast. Wordt een bepaalde schakelaar gesloten dan wordt de daarbij behorende regel doorverbonden met de kolom waarop de schakelaar is aangesloten. Het aftasten van de toetsen gaat als volgt:

Eerst wordt *alleen* bit 0 van register 56320 hoog gemaakt, waardoor spanning komt op de kolom 0. Nu wordt register 56321 gelezen. Is één van de schakelaars van de kolom 0 gesloten dan wordt één van de ingangspoorten van register 56321 'hoog'. Is geen enkele schakelaar gesloten dan is de inhoud van dit register 0 en wordt de uitgangspoort 1 van register 56320 hoog. Dit gaat zo door totdat kolom 7 hoog is geweest. Dit houdt in dat

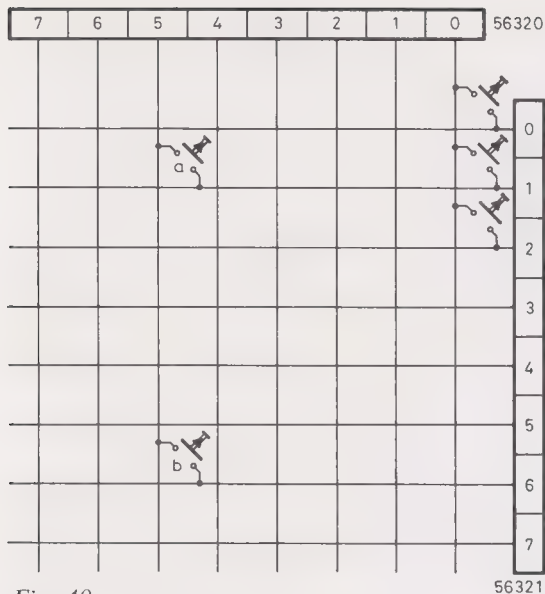


Fig. 40

register 56320 regelmatig na elkaar wordt ingeschreven met de getallen 1 (2^0), 2 (2^1), 4 (2^2), 8 (2^3), 16 (2^4), 32 (2^5), 64 (2^6) en 128 (2^7). Afhankelijk van de ingedrukte toets kan bij een bepaald getal in register 56320 een getal in register 56321 worden gelezen dat ook een macht van 2 is. Wordt schakelaar a in figuur 40 ingedrukt dan wordt het getal 2

(2^1) gelezen in register 56321 als het getal in register 56320 32 is. Was echter schakelaar b gesloten dan was in register 56321 het getal 64 te lezen. De inhoud van deze registers geven dus exact weer welke toets is ingedrukt. Dit aftasten gebeurt razend snel en wordt het 'Scannen' van het toetsenbord genoemd. Steeds als de subroutine SCNKEY wordt aangeroepen wordt het toetsenbord afgetast. Bij de normale werking van de computer gebeurt dat elke 1/60 sec. De subroutine vormt uit de inhoud van de registers een getal tussen 0 en 64 dat in de geheugenplaats 197 (\$00C5) wordt geschreven. Blijkt er *geen* toets ingedrukt dan is dit getal 64. Voor de andere getallen is in lijst 6 de toetsenaanduiding gegeven.

De toetsen CTRL, SHIFT en ⌘ worden niet gescand. Deze worden namelijk steeds in combinatie met een andere toets gebruikt. Afhankelijk van het getal in register 197 (lijst 6) en de toestand van de toetsen SHIFT, CTRL en ⌘ kan het ASCII getal worden bepaald dat in het toetsenbordbuffergeheugen wordt geschreven.

Verder valt nog te vermelden dat ook de toetsen RUN/STOP en RESTORE niet in het rooster zijn opgenomen.

De routine 'wacht op het indrukken van een toets' kan nu ook als volgt worden uitgevoerd:

Wacht op een toets in

49408 C100	A5 C5	LDA-Z,PAGE		Bepaal de inhoud van reg. 197.
49410 C102	C9 40	CMP-IMM	40	# 64 als geen toets is ingedrukt.
49412 C104	F0 FA	BEQ-REL	C100	Wacht op een toets.
49414 C106	20 E4FF	JSR-ABS	FFE4	Subr. GETIN, karakter uit de buffer.
49417 C109	20 D2FF	JSR-ABS	FFD2	Subr. CHROUT, print het karakter.
49420 C10C	60	RTS-IMPL		RETURN.

De werking van dit programma kunt u slechts waarnemen als u na SYS 49408 slechts zeer kort de RETURN toets aantipt!

Is een toets ingedrukt dan wordt de inhoud van register 197 dan pas weer \$ 40 als deze toets weer

wordt losgelaten. Dat maakt het mogelijk ook een routine 'wacht op het loslaten van de toets' te maken. Hiervoor kan het volgende programma worden gebruikt:

Wacht op een toets los

49408 C100	A5 C5	LDA-Z,PAGE		Bepaal de inhoud van reg. 197.
49410 C102	C9 40	CMP-IMM	40	Ga na of een toets is ingedrukt.
49412 C104	F0 FA	BEQ-REL	C100	Wacht op een toets.
49414 C106	20 E4FF	JSR-ABS	C100	GETIN routine.
49417 C109	20 D2FF	JSR-ABS	FFD2	CHROUT, print het karakter.
49420 C10C	A5 C5	LDA-Z,PAGE		Bepaal de inhoud van reg. 197.
49422 C10E	C9 40	CMP-IMM	40	Is de toets nog ingedrukt?
49424 C110	D0 FA	BNE-REL	C10C	Wacht tot hij wordt losgelaten.
49426 C112	60	RTS-IMPL		RETURN.

De eerste drie regels zijn voor het wachten op het indrukken van een toets. Daarna wordt het toetsenbordbuffergeheugen gelezen en het karakter geprint. Zolang een toets is ingedrukt zal de inhoud van het register 197 ongelijk zijn aan \$ 40 zodat in dat geval in regel 49424 steeds wordt teruggesprongen naar regel 49420, totdat de toets is losgelaten en de inhoud van register 197 weer gelijk is aan \$ 40. Dit laatste programmeel is de routine voor 'wacht op toets los'.

Ook dit programma moet gestart worden door na SYS 49408 de RETURN toets slechts zeer kort aan

te tippen. In principe hoort een programmeel 'wacht op toets in' steeds door een routine 'wacht op toets los' te worden *voorafgegaan*, dit voor het wachten op het loslaten van de RETURN toets na SYS 49408. Dit principe is dan ook in het volgende programma gehanteerd. Dit programma geeft een voorbeeld van het gebruik van de routines 'wacht op toets in' en 'wacht op toets los'. In dit geval wordt door het indrukken van een *bepaalde* toets (toets 3) een toon ten gehore gebracht, zolang de toets ingedrukt wordt gehouden. Figuur 41 geeft het stroomdiagram.

Wacht op een bepaalde toets in

49408	C100	A9 0F	LDA-IMM	0F	Voor maximaal volume.
49410	C102	8D 18D4	STA-ABS	D418	
49413	C105	A9 F0	LDA-IMM	F0	Voor Sustain/Release.
49415	C107	8D 06D4	STA-ABS	D406	
49418	C10A	A9 00	LDA-IMM	00	Voor Attack/Decay.
49420	C10C	8D 05D4	STA-ABS	D405	
49423	C10F	A9 11	LDA-IMM	11	Hoge byte frequentie.
49425	C111	8D 01D4	STA-ABS	D401	
49428	C114	A9 25	LDA-IMM	25	lage byte frequentie.
49430	C116	8D 08D4	STA-ABS	D400	
49433	C119	A5 C5	LDA-Z.PAGE		Bepaal de inhoud van re9. 197.
49435	C11B	C9 40	CMP-IMM	40	Is de toets nog in?
49437	C11D	D0 FA	BNE-REL	C119	Wacht op toets los.
49439	C11F	A9 10	LDA-IMM	10	Golfvorm triangel, Poort dicht.
49441	C121	8D 04D4	STA-ABS	D404	Zet geluid af.
49444	C124	A5 C5	LDA-Z.PAGE		Bepaal de inhoud van re9. 197.
49446	C126	C9 08	CMP-IMM	08	Toets 3 ingedrukt?
49448	C128	D0 FA	BNE-REL	C124	Wacht op toets 3 in.
49450	C12A	A9 11	LDA-IMM	11	Golfvorm triangel, Poort open.
49452	C12C	8D 04D4	STA-ABS	D404	Zet geluid aan.
49455	C12F	4C 19C1	JMP-ABS	C119	Naar wacht op toets los.

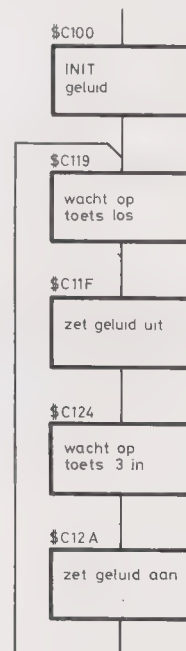


Fig. 41

Het initiëren van het geluid houdt in dat het volume, de vorm (ADSR) en de frequentie worden ingevoerd. Dit heeft plaats in de regels 49408 tot en met 49430. Hierna volgt de routine 'wacht op toets los'. Is er geen toets (meer) ingedrukt dan wordt het geluid, zo het er al was, uitgeschakeld door de poort (bit 0) van het betreffende geluidskanaal te sluiten (bit 0 = 0). Hierna volgt een programmeel waarin *niet* wordt gewacht totdat een *willekeurige* toets wordt ingedrukt, maar totdat toets 3 wordt ingedrukt. Dit wordt bereikt door de inhoud van register 197 te vergelijken met het getal 8 (zie lijst 6). Het programmeel 'wacht op toets 3' vindt u op de regels 49444 tot en met 49448. Is toets 3 ingedrukt dan wordt dit programmeel verlaten en het geluid ingeschakeld door de poort van het geluidskanaal (bit 0) te openen (bit 0 = 1). Hierna wordt teruggegaan naar de routine 'wacht

op toets los', waarna het geluid weer wordt uitgeschakeld als toets 3 wordt losgelaten.

Lijst 6. SCAN-getallen van het toetsenbord

nr. toets	nr. toets	nr. toets	nr. toets
0 INST/DEL	16 5	32 9	48 £
1 RETURN	17 R	33 I	49 *
2 CRSR hor.	18 D	34 J	50 ;
3 f7	19 6	35 0	51 CLR
4 f1	20 C	36 M	52
5 f3	21 F	37 K	53 =
6 f5	22 T	38 O	54
7 CRSR vert.	23 X	39 N	55 /
8 3	24 7	40 +	56 1
9 W	25 Y	41 P	57
10 A	26 G	42 L	58
11 4	27 8	43 -	59 2
12 Z	28 B	44 .	60 spatie
13 S	29 H	45 :	61
14 E	30 U	46 @	62 Q
15	31 V	47 ,	63

Selecteert u, door gebruik te maken van meerdere toetsen, verschillende frequenties dan is het mogelijk op deze manier van uw computer een orgeltje te maken.

6.4. Het printen van stringvariabelen

Voor het printen van stringvariabelen zijn er verschillende mogelijkheden die het betrekkelijk een-

voudig maken diverse teksten op het beeldscherm te schrijven. Met één van deze mogelijkheden hebben we al kennis gemaakt in het laatste programma van paragraaf 6.2. Het volgende programma is hier ongeveer identiek aan. Het belangrijkste verschil is dat het programma nu verlaten kan worden door het indrukken van de RETURN toets.

GET A\$

49408	C100	A2 00	LDX-IMM	00	Voor de lengte van de string.
49410	C102	8A	TXA-IMPL		Stringlengte in de ACCU.
49411	C103	48	PHA-IMPL		Save de stringlengte in de stack.
49412	C104	20 E4FF	JSR-ABS	FFE4	GETIN subroutine.
49415	C107	C9 00	CMP-IMM	00	Geen toets ingedrukt, S 00 in de ACCU.
49417	C109	F0 F9	BEQ-REL	C104	Wacht op een toets.
49419	C10B	20 D2FF	JSR-ABS	FFD2	CHROUT routine voor het Printen.
49422	C10E	A8	TYA-IMPL		Bewaar het karakter in het Y-register.
49423	C10F	68	PLA-IMPL		Haal de stringlengte weer op.
49424	C110	AA	TAX-IMPL		Stringlengte in het X-register.
49425	C111	98	TYA-IMPL		Karakter in de ACCU.
49426	C112	9D 00CA	STA-ABS,X	CA00	Karakter in de tabel op de Plaats X.
49429	C115	E8	INX-IMPL		Verhoog de stringlengte.
49430	C116	C9 0D	CMP-IMM	0D	Was de RETURN toets ingedrukt?
49432	C118	D0 E8	BNE-REL	C102	Zo nee, terug voor het volgend karakter.
49434	C11A	A9 00	LDA-IMM	00	\$ 00 in de ACCU.
49436	C11C	9D 00CA	STA-ABS,X	CA00	Voor het afsluiten van de tabel.
49439	C11F	60	RTS-IMPL		RETURN.

Start dit programma met:

PRINT CHR\$(147):SYS 49408.

Elk karakter dat u nu intoetst zal op uw scherm verschijnen, ook de grafische karakters en de diverse kleuren. Eveneens zal gereageerd worden op de toetsen voor de cursorbewegingen, hoewel de cursor niet te zien is. Ook RVS ON/OFF en CLR HOME zullen worden uitgevoerd. De subroutine die dit alles te werk stelt is CHROUT (regel 49419), die — zoals later zal blijken — niet alleen voor uitvoer naar het beeldscherm wordt toegepast. Deze routine wordt voorafgegaan door het programmadeel voor het invoeren van de karakters via het toetsenbord. Het programma is verder zo ingericht dat de ingetoetste karakters in de tabel worden opgeslagen (regels 49422 tot en met

49426). Na het verhogen van de stringlengte wordt nagegaan of de laatst ingedrukte toets de RETURN toets is. Is dat het geval (de ASCII code voor RETURN wordt ook in de tabel opgeslagen) dan wordt de tabel afgesloten met \$ 00 en wordt het programma verlaten.

Start het programma eens opnieuw en toets het volgende in (gevolgd door RETURN):

MICROCOMPUTER.

Bedenk dat de string vijftien karakters bevat (inclusief de punt en RETURN). Deze karakters worden over de karakters heen geschreven die zich eventueel al in de tabel bevonden, er wordt dus een nieuwe tabel gemaakt! We proberen nu het volgende programma:

PRINT A\$

49408	C100	A9 00	LDA-IMM	00	Laag byte van het tabeladres.
49410	C102	A0 CA	LDY-IMM	CA	Hoog byte van het tabeladres.
49412	C104	20 1EAB	JSR-ABS	AB1E	Subr. Print de string.
49415	C107	60	RTS-IMPL		RETURN.

Dit is de tweede mogelijkheid voor het printen van de string. Hierbij moet het aanvangsadres van de string in de volgorde lage byte - hoge byte in respectievelijk de accu en het Y-register worden geladen. De subroutine op adres \$ AB1E print alle karakters van de string, inclusief de RETURN. Deze subroutine gaat door totdat de geheugenplaats is bereikt die met \$ 00 is gevuld. Dan stopt hij het printen zodat u het woord 'MICROCOMPUTER.' op uw scherm ziet staan. U kunt ook de subroutine aanroepen met JMP \$ AB1E. De regel 49415 met RTS komt dan te vervallen. Het programma wordt dan verlaten met de RTS van de subroutine \$ AB1E.

Het derde programma in deze reeks geeft de mogelijkheid tot het printen van een file met een lengte van maximum 255 karakters.

De subroutine die hierbij wordt gebruikt (\$ F5C1, regel 49420) haalt zijn gegevens uit de geheugenplaats \$ 00B7 (voor de lengte van de file, maximaal 255 -\$ FF- tekens) en \$ 00BB en \$ 00BC voor respectievelijk de lage byte en de hoge byte van het file adres. Deze gegevens worden in de regels 49408 tot en met 49418 in deze plaatsen geladen. In dit geval is de file lengte gelijk aan 14 (\$ 0E).

PRINT FILE

49408 C100 A9 0E LDA-IMM	0E # 14, lengte van de file	naam
49410 C102 85 B7 STA-Z.PAGE	GeheugenPlaats FNLEN.	
49412 C104 A9 00 LDA-IMM	00 Lage byte van het file-adres	naam
49414 C106 85 BB STA-Z.PAGE	GeheugenPlaats FNADRL.	
49416 C108 A9 CA LDA-IMM	CA Hoge byte van het file-adres	naam
49418 C10A 85 BC STA-Z.PAGE	GeheugenPlaats FNADRH.	
49420 C10C 4C C1F5 JMP-ABS	F5C1	Subr. Print de file.

Start u dit programma met

PRINT CHR\$(147): SYS 49408

dan ziet u onze zin netjes op het scherm geprint!

6.5. Het invoeren van floating point variabelen

De moeilijkheid van het invoeren van floating point variabelen via het toetsenbord is dat de getallen niet in hun geheel worden ingevoerd maar cijfer voor cijfer, waarbij elk cijfer voor de computer niets meer is dan een teken in een karakterstring.

Voorwaarde voor het invoeren van een getal is dat na het intoetsen van het laatste cijfer de RETURN

toets wordt ingedrukt. Hiermee wordt de lengte van de string gemarkeerd. Uit deze string zal nu het getal moeten worden gevormd. Dit getal zal in een zodanige vorm in het geheugen moeten worden geplaatst dat de computer in staat zal zijn alle nodige bewerkingen ermee uit te voeren. Voor dit getal dient een plaats in de geheugenruimte te worden gereserveerd. In overeenstemming met BASIC kan deze plaats een *variabele* worden genoemd die met een bepaalde waarde (het getal) kan worden gevuld. Er zijn voor een floating point variabele vijf geheugenplaatsen nodig om een getal van de maximum grootte te kunnen bevatten. Het invoeren van een getal kan plaatsvinden met het volgende programma:

INPUT A

49408 C100 A2 FF LDX-IMM	FF # -1 voor de stringlengte.
49410 C102 E8 INX-IMPL	Verhoog de stringlengte.
49411 C103 20 CFFF JSR-ABS	FFCF CHRIN routine voor input.
49414 C106 9D 00C8 STA-ABS,X	C800 Karakter in Plaats X van de buffer.
49417 C109 C9 0D CMP-IMM	0D Einde van de string (RETURN toets)?
49419 C10B D0 F5 BNE-REL	C102 Zo niet, terug voor volgend karakter.
49421 C10D 20 D2FF JSR-ABS	FFD2 Cursor naar de volgende regel.
49424 C110 A9 00 LDA-IMM	00 Lage byte bufferadres.
49426 C112 85 22 STA-Z.PAGE	Pointer lage byte.
49428 C114 A9 C8 LDA-IMM	C8 Hoge byte bufferadres.
49430 C116 85 23 STA-Z.PAGE	Pointer hoge byte.
49432 C118 8A TXA-IMPL	Lengte van de string in de ACCU.
49433 C119 20 B5B7 JSR-ABS	B7B5 Omzetting naar floating point in FPAC.

49436	C11C	A2 00	LDX-IMM	00	Lage byte van het adres voor de variab.
49438	C11E	A0 C9	LDY-IMM	C9	Hoge byte van het adres voor de variab.
49440	C120	20 D4BB	JSR-ABS	BBD4	Variabele uit FPAC in toegewezen Plaats.
49443	C123	60	RTS-IMPL		RETURN.

Het stroomdiagram geeft figuur 42. Hoewel met de GETIN routine ook de afzonderlijke cijfers (karakters) kunnen worden ingevoerd is hier gekozen voor het CHRIN routine. Deze subroutine verwezenlijkt het BASIC INPUT statement.

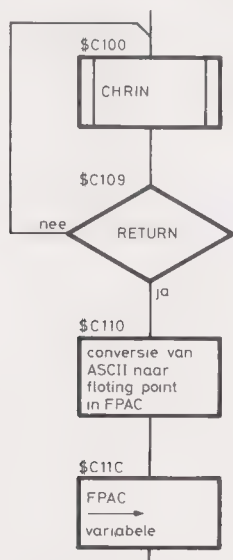


Fig. 42

Na het starten van het programma met SYS 49408 wordt eerst het X-register met \$ FF gevuld. Dit komt overeen met een -1 decimaal. Direct daarna wordt de inhoud van dit register opgehoogd tot \$ 00. Nu komt de subroutine CHRIN in werking. Hierdoor wordt de cursor aangezet. Elk karakter dat nu wordt ingetoetst wordt zichtbaar gemaakt op het scherm en geplaatst in de input buffer (geheugenplaatsen \$ 0200 - \$ 0258). Deze buffer kan maximaal 88 tekens bevatten. Elk karakter wordt geprint op de plaats die door de cursor wordt aan-

gegeven. Dit gaat zo door totdat op de RETURN toets wordt gedrukt. Nu haalt de CHRIN routine één karakter uit de inputbuffer. Dit karakter vinden we in de accu. In regel 49414 wordt dit karakter in geheugenplaats \$ C800+X geplaatst. Voor het volgende karakter wordt terug gegaan naar regel 49410 en na het verhogen van de stringlengte in register X wordt opnieuw de subroutine CHRIN aangeroepen die het volgende karakter uit de inputbuffer ophaalt (first in, first out). Dit gaat zo door tot het laatste karakter, dat voor RETURN, is opgehaald. Nu wordt de string omgezet naar een getal (van ASCII naar floating point). Dit doet de subroutine in regel 49433 op geheugenplaats \$ B7B5. Het adres van de string moet daarvoor eerst in de geheugenplaatsen \$ 0022 en \$ 0023 worden geladen (pointer) en de stringlengte in de accu (TXA, regel 49432). De subroutine vormt uit de afzonderlijke cijfers het getal en plaatst dit in de geheugenplaatsen floating point accu (FPAC). Bij alle bewerkingen met floatingpoint variabelen worden *deze* geheugenplaatsen gebruikt. Nu moet het getal nog in het juiste adres van de variabele worden opgeslagen. Dit verzorgt de subroutines \$ BBD4 op regel 49440. Het variabele adres wordt daarvoor in het X-register (lage byte) en in het Y-register (hoge byte) geladen (regels 49436 en 49438). Voor het invoeren van een volgend getal moet de routine opnieuw worden aangeroepen (vanuit BASIC met SYS 49408). Het adres voor deze nieuwe variabele moet dan vijf plaatsen hoger worden gekozen als die van de vorige.

Uiteraard kan een getal dat in het geheugen is opgeslagen ook op het beeldscherm worden geprint. Nu moet de omgekeerde werking plaatsvinden:

PRINT A

49408	C100	A9 00	LDA-IMM	00	Lage byte van het adres van de variab.
49410	C102	A0 C9	LDY-IMM	C9	Hoge byte van het adres van de variab.
49412	C104	20 A2BB	JSR-ABS	BBA2	Variabele in FPAC.
49415	C107	20 DDBD	JSR-ABS	BDDD	Van floating point naar ASCII string.
49418	C10A	A2 00	LDX-IMM	00	Voor de lengte van de string.
49420	C10C	BD 0001	LDA-ABS,X	0100	Karakter in de ACCU.
49423	C10F	F0 06	BEQ-REL	C117	Einde van de string?
49425	C111	20 D2FF	JSR-ABS	FFD2	Print het karakter.
49428	C114	E8	INX-IMPL		Verhoog de stringlengte.
49429	C115	D0 F5	BNE-REL	C10C	Terug voor volgend karakter.
49431	C117	A9 0D	LDA-IMM	0D	Einde string, cursor naar nieuwe regel.
49433	C119	20 D2FF	JSR-ABS	FFD2	CHROUT routine.
49436	C11C	60	RTS-IMPL		RETURN.

Eerst wordt het getal uit het variabele adres opgehaald door de subroutine \$ BBA2 op regel 49412. Hiervoor moet dat adres in de accu (lage byte) en in het Y-register (hoge byte) worden geladen. Het getal wordt in FPAC geplaatst. Nu heeft de omzetting van floating point naar de ASCII string plaats (subroutine \$ BDDD). Deze string vinden we op het adres aanvangend met \$0100. Hiervoor wordt dus een gedeelte van de (onderste) stack ruimte in beslag genomen. De string is afgesloten met \$ 00.

De regels 49418 tot en met 49429 zorgen voor het printen van de string. Het printen wordt afgesloten met RETURN (regels 49431 en 49433). Roept u dit programma aan met SYS 49408 dan wordt het getal geprint dat u met het voorgaande programma hebt ingevoerd.

Uitgaande van hetgeen hiervoor is beschreven laat zich op eenvoudige wijze een inputroutine samenstellen:

INPUT routine

49408 C100 A9 3F LDA-IMM	3F	# 63, voor ASCII "?".
49410 C102 20 D2FF JSR-ABS	FFD2	CHROUT voor het Printen van "?".
49413 C105 A2 FF LDX-IMM	FF	# -1, voor de stringlength.
49415 C107 E8 INX-IMPL		Verhoog de stringlength.
49416 C108 20 CFFF JSR-ABS	FFCF	CHRIN routine voor input.
49419 C10B 9D 00C8 STA-ABS,X	C800	Karakter in Plaats X van de buffer.
49422 C10E C9 0D CMP-IMM	0D	Einde van de string (RETURN toets)?
49424 C110 D0 F5 PNE-REL	C107	Zo niet, terug voor volgend karakter.
49426 C112 20 D2FF JSR-ABS	FFD2	Cursor naar de volgende regel.
49429 C115 A9 00 LDA-IMM	00	Lage byte bufferadres.
49431 C117 85 22 STA-Z.PAGE		Pointer lage byte.
49433 C119 A9 C8 LDA-IMM	C8	Hoge byte bufferadres.
49435 C11B 85 23 STA-Z.PAGE		Pointer hoge byte.
49437 C11D 8A TXA-IMPL		Length van de string in de ACCU.
49438 C11E 20 B5B7 JSR-ABS	B7B5	Van ASCII naar floating Point in FPAC.
49441 C121 60 RTS-IMPL		RETURN.

Deze inputroutine kan als subroutine in een programma worden toegepast. Eerst wordt een vraagteken op het beeldscherm geprint, zoals dat bij een BASIC INPUT statement gebruikelijk is.

De ASCII code voor '?' is 63 (\$ 3F). Het vraagteken wordt door de subroutine CHROUT op het scherm geprint. Daarna wordt het getal ingevoerd met de subroutine CHRIN en in een string in de buffer geplaatst. Ten slotte vinden we het ingevoerde getal als floating point in FPAC. Deze su-

broutine plaatst het getal niet op het variabele adres. Indien nodig zal dit in het hoofdprogramma moeten gebeuren.

6.6. Het laden van geheugenplaatsen en het printen van hun inhoud

Het laden van een geheugenplaats met een getal komt overeen met het BASIC statement POKE X, Y. Het volgende programma kan hiervoor worden gebruikt:

POKE X, Y

49408 C100 A9 58 LDA-IMM	58	# 88, ASCII voor "X".
49410 C102 20 D2FF JSR-ABS	FFD2	CHROUT voor het Printen van "X".
49413 C105 20 00C2 JSR-ABS	C200	Subr. INPUT voor INPUT X.
49416 C108 20 F7B7 JSR-ABS	B7F7	INT(FPAC) in \$ 0014 en \$ 0015.
49419 C10B A9 59 LDA-IMM	59	# 89, voor ASCII "Y".
49421 C10D 20 D2FF JSR-ABS	FFD2	CHROUT voor het Printen van "Y".
49424 C110 20 00C2 JSR-ABS	C200	Subr. INPUT voor INPUT Y.
49427 C113 20 9BBC JSR-ABS	BC9B	INT(FPAC) in FPAC
49430 C116 A5 64 LDA-Z.PAGE		Voor hoge byte van Y.
49432 C118 D0 07 BNE-REL	C121	Sprong als Y>255.
49434 C11A A5 65 LDA-Z.PAGE		Voor lage byte van Y.
49436 C11C A0 00 LDY-IMM	00	Waarde van Y schrijven
49438 C11E 91 14 STA-(IND),Y		in het adres X.
49440 C120 60 RTS-IMPL		RETURN.
49441 C121 20 48B2 JSR-ABS	B248	PRINT "ILLEGAL QUANTITY ERROR".

49444	C124	60	RTS-IMPL		RETURN.
49664	C200	A9 3F	LDA-IMM	3F	SUBROUTINE INPUT.
49666	C202	20 D2FF	JSR-ABS	FFD2	
49669	C205	A2 FF	LDX-IMM	FF	
49671	C207	E8	INX-IMPL		
49672	C208	20 CFFF	JSR-ABS	FFCF	
49675	C20B	9D 00C8	STA-ABS,X	C800	
49678	C20E	C9 0D	CMP-IMM	0D	
49680	C210	D0 F5	BNE-REL	C207	
49682	C212	20 D2FF	JSR-ABS	FFD2	
49685	C215	A9 00	LDA-IMM	00	
49687	C217	85 22	STA-Z.PAGE		
49689	C219	A9 C8	LDA-IMM	C8	
49691	C21B	85 23	STA-Z.PAGE		
49693	C21D	8A	TXA-IMPL		
49694	C21E	20 B5B7	JSR-ABS	B7B5	
49697	C221	60	RTS-IMPL		RETURN.

Ook nu is de moeilijkheid dat de karakters in een string worden ingevoerd. De input routine uit de vorige paragraaf is dan ook nodig om het getal in FPAC in te voeren. Eerst doen we dit voor het getal X, dat het adres is van de geheugenplaats waar het getal Y moet worden geschreven. Eerst wordt door de regels 49408 en 49410 het karakter 'X' op het scherm geprint. Door het aanroepen van de subroutine INPUT verschijnt hierna het vraagteken en de cursor. Nu kan het getal voor X worden ingetoetst en na RETURN vinden we dit in FPAC. Hoewel we voor een adres van een geheugenplaats steeds een heel getal zullen intoetsen passen we toch hiervoor de subroutine op regel 49416 toe. Deze bepaalt de integer van het getal in FPAC. Het belangrijkste is echter dat het resultaat hiervan wordt geplaatst in de geheugenplaatsen \$ 0014 (lage byte) en \$ 0015 (hoge byte). Verder zorgt deze subroutine er voor dat bij een getal groter dan 65535 de foutmelding 'ILLEGAL QUANTITY

ERROR' wordt gegeven. Hierna is het mogelijk het getal Y in te voeren. Om dit in de juiste vorm te krijgen passen we subroutine \$ BC9B in regel 49427 toe. Deze bepaalt de integer van het getal in FPAC en plaatst het resultaat weer terug in FPAC. Is een getal groter dan 255 ingevoerd dan bestaat het uit een hoge byte in geheugenplaats \$ 0064 van FPAC en een lage byte in geheugenplaats \$ 0065. Een getal groter dan 255 kan niet in een geheugenplaats worden geschreven, vandaar dat een foutmelding moet volgen. Dit verzorgt de subroutine \$ B248 in regel 49441. Is het getal kleiner dan 256 dan wordt dit in regel 49438 opgehaald uit FPAC zodat het in regel 49438 op het adres kan worden geplaatst dat hiervoor in de geheugenplaatsen \$ 0014 en \$ 0015 is opgeslagen. Het printen van de inhoud van een geheugenplaats komt overeen met het BASIC statement PRINT PEEK (X). Gebruik hiervoor het volgende programma.

PRINT PEEK (X)

49408	C100	A9 58	LDA-IMM	58	# 88, voor ASCII "X".
49410	C102	20 D2FF	JSR-ABS	FFD2	CHROUT voor het Printen van "X".
49413	C105	20 00C2	JSR-ABS	C200	Subr. INPUT voor INPUT X.
49416	C108	20 00B8	JSR-ABS	B80D	PEEK(X) in FPAC.
49419	C10B	20 DDBD	JSR-ABS	BDDD	Van floating Point naar ASCII string.
49422	C10E	A2 00	LDX-IMM	00	Voor de lengte van de string.
49424	C110	BD 0001	LDA-ABS,X	0100	Karakter in de ACCU.
49427	C113	F0 06	BEQ-REL	C11B	Einde van de string?
49429	C115	20 D2FF	JSR-ABS	FFD2	Print het karakter.
49432	C118	E8	INX-IMPL		Verhoog de stringlengte.
49433	C119	D0 F5	BNE-REL	C110	Terug voor het volgende karakter.
49435	C11B	A9 0D	LDA-IMM	0D	Einde string, cursor naar nieuwe regel.
49437	C11D	20 D2FF	JSR-ABS	FFD2	CHROUT routine.
49440	C120	60	RTS-IMPL		RETURN.
49664	C200	A9 3F	LDA-IMM	3F	SUBROUTINE INPUT.
49666	C202	20 D2FF	JSR-ABS	FFD2	

49669	C205	A2 FF	LDX-IMM	FF	
49671	C207	E8	INX-IMPL		
49672	C208	20 CFFF	JSR-ABS	FFCF	
49675	C20B	9D 00C8	STA-ABS,X	C800	
49678	C20E	C9 0D	CMP-IMM	0D	
49680	C210	D0 F5	BNE-REL	C207	
49682	C212	20 D2FF	JSR-ABS	FFD2	
49685	C215	A9 00	LDA-IMM	00	
49687	C217	85 22	STA-Z.PAGE		
49689	C219	A9 C8	LDA-IMM	C8	
49691	C21B	85 23	STA-Z.PAGE		
49693	C21D	8A	TXA-IMPL		
49694	C21E	20 B5B7	JSR-ABS	B7B5	
49697	C221	60	RTS-IMPL		RETURN.

Eerst wordt op de bekende wijze het adres X van de geheugenplaats ingevoerd. Daarna kan direct met de subroutine \$ BBOD in regel 49416 de inhoud van deze geheugenplaats in FPAC worden geplaatst. Heeft u een groter getal dan 65535 ingevoerd dan volgt een foutmelding. Nu kunt u eventueel de inhoud van FPAC op het adres van een

variabele laten plaatsen met de subroutine \$ BBD4, zoals dat in paragraaf 6.5 is getoond. Hier is dat niet gedaan. Direct na de PEEK routine wordt de inhoud van FPAC omgezet in een ASCII string (regel 49419) die in de regels 49422 tot en met 49433 op het beeldscherm wordt geprint.

7. SAVEN, LOADEN en PRINTEN

7.1. Het SAVEN en LOADEN van programma's

Het SAVEN van een programma naar een cassette-recorder kan met een BASIC commando

SAVE "(naam)"

Compleet is dit commando niet, de computer vult het aan tot

SAVE "(naam)",01,00

Het eerste getal dat is toegevoegd is het device nummer, voor de tape \$ 01, en het tweede getal is het 'commando' of 'secondairy adres'. Voor het saven van een programma met een machinetaalprogramma moeten vooraf de volgende gegevens worden ingevoerd:

- Een filennummer. Het saven van een programma gaat met het openen van een kanaal gepaard waarvoor een filennummer nodig is. Dit kan een nummer zijn van \$ 00 tot en met \$ FF.
- Het device nummer. Dit is voor de cassette-recorder \$ 01 en voor de disk drive \$ 08.
- Het commando. Dit commando is enigszins afhankelijk van het programma dat later voor loa-

den wordt gebruikt. Is het getal hiervoor \$ 00 dan wordt later bij het laden het programma in de BASIC programmaruimte geladen, te beginnen op adres f 0800, tenzij in het LOAD programma een startadres is ingevoerd. Is het getal voor het commando \$ 01 dan wordt later bij het loaden het programma in dezelfde geheugen-plaatsen teruggeplaatst als die waar het zich bevond bij het saven, ook indien in het LOAD programma geen startadres is opgenomen.

- Het startadres van het programma. Als het startadres van het programma niet het gebruikelijke BASIC startadres \$ 0800 is dan moet het juiste startadres worden ingevoerd.
- Het eindadres van het programma. Als het eindadres zich niet bevindt in de geheugen-plaatsen \$ 002D en \$ 002E (eind van het BASIC programma, zie paragraaf 5.2) dan moet het worden ingevoerd.

Het resultaat van deze punten zal het volgende programma opleveren:

SAVE een programma

49408	C100	A9 01	LDA-IMM	01	File nummer.
49410	C102	A2 01	LDX-IMM	01	Device nr. \$ 01 tape, \$ 08 disk drive.
49412	C104	A0 00	LDY-IMM	00	Voor secondair adres.
49414	C106	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de SAVE routine.
49417	C109	A9 09	LDA-IMM	09	Lengete van de naam (SAVE/LOAD).
49419	C10B	A2 00	LDX-IMM	00	La9e byte tabeladres (naam in de tabel).
49421	C10D	A0 CA	LDY-IMM	CA	Hoge byte tabeladres.
49423	C10F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49426	C112	A9 00	LDA-IMM	00	La9e byte startadres.
49428	C114	85 14	STA-Z.PAGE		Vector la9e byte.
49430	C116	A9 C1	LDA-IMM	C1	Hoge byte startadres.
49432	C118	85 15	STA-Z.PAGE		Vector hoge byte.
49434	C11A	A9 14	LDA-IMM	14	Adres van de vector.
49436	C11C	A2 24	LDX-IMM	24	La9e byte eindadres+1.
49438	C11E	A0 C1	LDY-IMM	C1	Hoge byte eindadres+1.
49440	C120	20 D8FF	JSR-ABS	FFD8	SAVE routine.
49443	C123	60	RTS-IMPL		RETURN.

Het filennummer, device nummer en het commando wordt door de subroutine SETLFS ingevoerd in regel 49414. Het filennummer is \$ 01 en ook het devicenummer is \$ 01 zodat de cassette-recorder wordt gebruikt voor het saven. Het secundair adres in \$ 00. De LOAD routine zal er dus voor moeten

zorgen dat het naar de tape geschreven programma weer op de juiste plaats wordt terug gebracht. Overigens heeft \$ 00 als secundair adres hetzelfde effect als \$ FF. De naam van het te saven programma dient u eerst in te voeren in de tabel (\$ CA00) met bijvoorbeeld het GET A\$ program-

ma. Als voorbeeld is SAVE/LOAD als programmaam gekozen.

Het adres waar deze naam begint (de tabel) wordt ingevoerd door de subroutine SETNAM op regel 49423. Nu wordt het startadres in de geheugenplaatsen \$ 0014 en \$ 0015 geladen. Om niet apart een programma te moeten invoeren om dit SAVE programma te kunnen proberen is hiervoor het startadres \$ C100 gekozen, zodat het SAVE programma zelf naar de tape wordt geschreven. De plaats waar deze vector is te vinden (\$ 0014) wordt in de accu geplaatst (regel 49434). De SAVE subroutine op regel 49440 zorgt er uiteindelijk voor dat het programma op de band komt.

Er is slechts een kleine verandering nodig om het programma voor de disk drive geschikt te maken. Hiervoor hoeft slechts het devicenummer \$ 01 veranderd te worden in \$ 08 (regel 49410). Verder blijft het programma hetzelfde.

Voor het laden van een machinetaalprogramma moeten de volgende gegevens worden ingevoerd:

- Een filenummer. Dit kan een waarde hebben van \$ 00 tot en met \$ FF.
- Een devicenummer.
- Geen secundair adres (\$ 00 of \$FF).
- Zonodig het startadres van het programma. Indien bij de SAVE routine een commando \$ 01 is gegeven dan behoeft geen startadres in de LOAD routine te worden ingevoerd. Het programma wordt dan automatisch in de oorspronkelijke geheugenplaatsen geladen. Is dat niet het geval dan moet het startadres worden ingevoerd in de LOAD routine. Wordt dan geen startadres ingevoerd dan wordt het programma in de BASIC programmaruimte geladen, te beginnen met geheugenplaats \$ 0800.
- Een LOAD routine kan ook worden gebruikt voor het verifiëren van een programma op de band. Daarom moet een teken (flag) worden gezet voor LOAD (flag = \$ 00) of voor VERIFY (flag = \$ 01).

Een load routine geeft het volgende programma:

LOAD een programma

49664 C200	A9 01	LDA-IMM	01	File nummer.
49666 C202	A2 01	LDX-IMM	01	Device nr. \$ 01 tape, \$ 08 disk drive.
49668 C204	A0 00	LDY-IMM	00	Voor secundair adres.
49670 C206	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor LOAD routine.
49673 C209	A9 09	LDA-IMM	09	Lengte van de naam (SAVE/LOAD).
49675 C20B	A2 00	LDX-IMM	00	Lage byte tabeladres (naam in de tabel).
49677 C20D	A0 CA	LDY-IMM	CA	Hoge byte tabeladres.
49679 C20F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49682 C212	A9 00	LDA-IMM	00	Flag, \$00 voor LOAD, \$ 01 voor VERIFY.
49684 C214	A2 00	LDX-IMM	00	Lage byte startadres.
49686 C216	A0 C1	LDY-IMM	C1	Hoge byte startadres.
49688 C218	20 D3FF	JSR-ABS	FFD5	LOAD routine.
49691 C21B	60	RTS-IMPL		RETURN.

Om dit programma (en de SAVE routine) te kunnen proberen is bij uitzondering gekozen voor het startadres \$ C200. De eerste acht regels van het programma zijn gelijk aan die van de SAVE routine. In regel 49682 wordt de flag voor LOAD ingevoerd. Daarna worden de X- en Y-registers met het startadres geladen, waarna de LOAD routine het programma van de band haalt. Wilt u geen startadres invoeren dan dient u het X- en het Y-register te vullen met \$ FF.

Moet het programma worden gebruikt voor het verifiëren dan moet de accu in regel 49682 met \$ 01 worden gevuld. Voor wat betreft het adres van het te verifiëren programma in het geheugen geldt hetzelfde als bij de normale LOAD functie.

Ook nu is het programma eenvoudig geschikt te maken voor de disk drive, \$ 01 in regel 49666 moet dan worden veranderd in \$ 08.

7.2. Het SAVEN en LOADEN van een file naar de disk

Het saven van een file naar de disk verloopt anders dan het saven van een programma. De karakters waaruit de file wordt opgebouwd moeten één voor één naar de disk drive worden verzonden. Vooraf moeten deze karakters in de geheugenruimte aanwezig zijn. Ze kunnen met de GET A\$ routine in de tabel worden ingevoerd. Omdat ook een naam voor de file in de tabel moet worden geplaatst (op \$ CA00) laten we de karakters voor de file aanvan-

gen op \$ CA10. Analooq aan het BASIC statement moet ook nu een file geopend worden (bijvoorbeeld: OPEN 2,8,2) en later weer worden gesloten.

Met de volgende punten moet het programma worden opgebouwd:

- Een filennummer. Dit kan een nummer zijn van \$ 00 tot en met \$ 7F (127).
- Het devicennummer. Normaal voor de disk \$ 08.
- Een derde nummer. Dit is in dit geval een kanaalnummer en kan van \$ 02 tot en met \$ 0E (14) zijn. \$ 00 en \$ 01 worden gebruikt bij het saven en loaden van programma's en \$ 0F (15) bij commando's.
- Het invoeren van de naam. Bij het saven van een file moet de naam de volgende gegevens bevatten:
 - Het nummer van de disk drive. Als u met

twee drives werkt dan moet het nummer 0: of het nummer 1: voor de betreffende file in de naam geplaatst worden.

2. De naam, bijvoorbeeld SAVE/LOAD.

3. De soort file, in dit geval sequential (S).

4. Een W voor het schrijven (WRITE) naar de drive.

De volledige naam zou daarom kunnen zijn: "@0:SAVE/LOAD,S,W".

Het teken @ wordt gebruikt als een bestaande file moet worden 'overschreven'.

e. Het openen van een kanaal, zoals bij BASIC OPEN 2,8,2.

f. Voor het saven moet de file uitgang worden.

g. Elk karakter moet worden opgehaald uit de tabel om te worden verstuurd naar de disk drive.

h. Het kanaal moet weer worden gesloten.

Het volgende programma is hiervan het resultaat:

SAVE een file naar de disk

49408 C100	A9 02	LDA-IMM	02	File nummer.
49410 C102	A2 08	LDX-IMM	08	Device nummer, \$ 08 voor disk drive.
49412 C104	A0 02	LDY-IMM	02	Secondair adres.
49414 C106	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de file.
49417 C109	A9 0D	LDA-IMM	0D	Lengte van de naam (SAVE/LOAD,S,W).
49419 C10B	A2 00	LDX-IMM	00	Lage byte tabel adres (naam in tabel).
49421 C10D	A0 CA	LDY-IMM	CA	Hoge byte tabeladres.
49423 C10F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49426 C112	20 C0FF	JSR-ABS	FFC0	OPEN, open de file.
49429 C115	A2 02	LDX-IMM	02	File nummer.
49431 C117	20 C9FF	JSR-ABS	FFC9	CHKOUT, Kanaal wordt uitgang.
49434 C11A	A0 00	LDY-IMM	00	Lengte van de file.
49436 C11C	B9 10CA	LDA-ABS,Y	CA10	Karakter in de ACCU.
49439 C11F	F0 06	BEQ-REL	C127	Spring bij einde van de file.
49441 C121	20 D2FF	JSR-ABS	FFD2	Karakter naar de disk drive.
49444 C124	C8	INY-IMPL		Verhoog de filelengte.
49445 C125	D0 F5	BNE-REL	C11C	Terug naar volgend karakter.
49447 C127	A9 02	LDA-IMM	02	File nummer.
49449 C129	20 C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49452 C12C	20 E7FF	JSR-ABS	FFE7	CLALL.
49455 C12F	60	RTS-IMPL		RETURN.

Het invoeren van de gegevens voor de file kennen we al van de vorige programma's (regel 49408 tot en met 49423). In regel 49426 wordt het kanaal geopend door de subroutine OPEN op \$ FFC0. De betreffende file (filennummer 2, regel 49429) wordt door de subroutine CHKOUT op \$ FFC9 uitgang. In regel 49436 wordt een karakter uit de tabel gehaald en in regel 49441 door de subroutine CHROUT naar de disk drive gestuurd. Er is vanuit gegaan dat de karakters in de tabel zijn geplaatst met de GET A\$ routine. In dat geval is de file afgesloten met \$ 00 (het laatste karakter is dat voor

RETURN). Is de accu met deze waarde geladen dan stopt het programma door het sluiten van het kanaal (regels 49447 tot en met 49452) met de subroutine CLOSE op \$ FFC3 en CLALL op \$ FFE7.

Een programma voor het loaden van een file vertoont veel overeenkomst met het voorafgaande. De verschillen zijn:

- De naam moet inplaats van een W voor WRITE een R voor READ bevatten: "SAVE/LOAD,S,R". Het nummer van de disk drive (0

- of 1) kan nodig zijn (bij twee drives), maar het teken @ moet achterwege blijven.
- b. De geopende file moet een ingang worden.

- c. De karakters moeten worden ingelezen van de drive en worden geschreven in de tabel.
- In het volgende programma is dat verwezenlijkt:

LOAD een file van de disk

49408	C100	A9 02	LDA-IMM	02	File nummer.
49410	C102	A2 08	LDX-IMM	08	Device nummer, \$ 08 voor disk drive.
49412	C104	A0 02	LDY-IMM	02	Secondair adres.
49414	C106	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de file.
49417	C109	A9 0D	LDA-IMM	0D	Lengte van de naam (SAVE/LOAD,S,R).
49419	C10B	A2 00	LDX-IMM	00	Laag byte tabeladres (naam in tabel).
49421	C10D	A0 0A	LDY-IMM	0A	Hoge byte tabeladres.
49423	C10F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49426	C112	20 C0FF	JSR-ABS	FFC0	OPEN, open de file.
49429	C115	A2 02	LDX-IMM	02	File nummer.
49431	C117	20 C6FF	JSR-ABS	FFC6	CHKIN, kanaal wordt ingang.
49434	C11A	A0 00	LDY-IMM	00	Lengte van de file.
49436	C11C	20 CFFF	JSR-ABS	FFCF	CHRIN, invoer van een karakter.
49439	C11F	99 10CA	STA-ABS,Y	CA10	Karakter in Plaats Y van de tabel.
49442	C122	C9 0D	CMP-IMM	0D	Laatste karakter van de file?
49444	C124	F0 03	BEQ-REL	C129	Spring bij einde file.
49446	C126	C8	INY-IMPL		Verhoog de filelengte.
49447	C127	D0 F3	BNE-REL	C11C	Terug voor volgend karakter.
49449	C129	A9 02	LDA-IMM	02	Einde file, file nummer.
49451	C12B	20 C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49454	C12E	20 E7FF	JSR-ABS	FFE7	CLALL.
49457	C131	60	RTS-IMPL		RETURN.

Uiteraard moet de juiste naam in de tabel weer worden gevormd met de GET A\$ routine. In regel 49431 wordt het kanaal een ingang door het toepassen van de CHKIN routine op \$ FFC6. Het inlezen geschiedt karakter voor karakter met de CHRIN routine op \$ FFCF (regel 49463) die met de STA instructie in de tabel wordt geschreven. Het laatste karakter voor de file is dat voor RETURN (\$OD). Hiermee wordt het invoeren beëindigd en het kanaal gesloten.

7.3. Het saven en loaden van een file naar de tape

Het saven van een file naar de tape verschilt belangrijk van het saven van een file naar de disk. Bij het laatste behoeven slechts de karakters aan de disk drive te worden aangeboden. Het eigenlijke schrijven naar de schijf wordt door de drive zelf verzorgd. Voor het schrijven naar de tape is echter in de computer zelf een programma aanwezig die daarvoor de karakters uit de tape I/O buffer haalt. Deze karakters moeten vóór het saven daarin worden overgebracht vanuit de tabel. Is de tape I/O buffer vol dan gaat het programma automatisch, dus zonder daarvoor een commando nodig te hebben, over op het schrijven naar de tape. Dit schrijven gebeurt twee keer na elkaar zodat twee dezelfde files na elkaar op de band staan en later bij het

loaden eventuele fouten kunnen worden opgemerkt. De tape I/O buffer kan niet meer dan 191 karakters bevatten zodat de filelengte hierdoor is beperkt tot 191 karakters. De werkwijze is nu als volgt:

Met de GET A\$ routine brengen we eerst de naam in de tabel (gewoon SAVE/LOAD) op \$ CA00 en daarna een aantal karakters voor de file op \$ CA10 en verder. Uiteraard kan het voorkomen dat de file *korter* is dan 191 tekens. Dit moet dan in het programma verwerkt worden zodat in elk geval wordt overgegaan tot het saven van de file.

Het programma moet de volgende punten bevatten:

- Het filenummer.
- Het devicenummer (voor de tape \$ 01).
- Het secondair adres. Dit moet bij het *saven* naar de tape van een file \$ 01 zijn.
- De naam. Deze bevat verder geen gegevens (SAVE/LOAD).
- Het openen van het kanaal.
- De file tot 'uitgang' maken.
- Het vullen van de tape I/O buffer met de file.
- Het saven van de file.
- Het sluiten van het kanaal.

Het volgende programma voldoet hieraan:

SAVE een file naar de tape

49408	C100	A9 01	LDA-IMM	01	File nummer.
49410	C102	A2 01	LDX-IMM	01	Device nummer, \$ 01 voor tape.
49412	C104	A0 01	LDY-IMM	01	Secondair adres, voor save file.
49414	C106	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de file.
49417	C109	A9 09	LDA-IMM	09	Lengthe van de naam (SAVE/LOAD).
49419	C10B	A2 00	LDX-IMM	00	Laagste byte tabeladres (naam in tabel).
49421	C10D	A0 CA	LDY-IMM	CA	Hogste byte tabeladres.
49423	C10F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49426	C112	20 C0FF	JSR-ABS	FFC0	OPEN, open de file.
49429	C115	A2 01	LDX-IMM	01	File nummer.
49431	C117	20 C9FF	JSR-ABS	FFC9	CHKOUT, kanaal wordt uitgegaan.
49434	C11A	A0 FF	LDY-IMM	FF	# -1, voor de lengthe van de file.
49436	C11C	C8	INY-IMPL		Verhoog de filelengthe.
49437	C11D	B9 10CA	LDA-ABS,Y	CA10	Karakter in de ACCU.
49440	C120	20 D2FF	JSR-ABS	FFD2	Karakter in de tape I/O buffer.
49443	C123	C9 0D	CMF-IMM	0D	Laatste karakter?
49445	C125	D0 F5	BNE-REL	C11C	Terug voor volgend karakter.
49447	C127	A9 BF	LDA-IMM	BF	Einde file. # 191 voor volle buffer.
49449	C129	85 A6	STA-Z,PAGE		Buffer Pointer in de hoogste stand.
49451	C12B	20 D2FF	JSR-ABS	FFD2	Tape I/O buffer naar de tape.
49454	C12E	A9 01	LDA-IMM	01	File nummer.
49456	C130	20 C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49459	C133	20 E7FF	JSR-ABS	FFE7	CLALL.
49462	C136	60	RTS-IMPL		RETURN.

Ten opzichte van het programma voor het save van een file naar de disk is het devicenummer anders (\$ 01), evenals het secondair adres (\$ 01 voor save). De kern van het programma vormen de regels 49434 tot en met 49451. Eerst wordt de karakterteller \$ 00 (regels 49434 en 49436). Daarna wordt een karakter uit de tabel in de accu geladen. De subroutine CHROUT op \$ FFD2 (regel 49440) zorgt ervoor dat het karakter in de tape I/O buffer wordt geplaatst. Dit wordt karakter voor karakter gedaan en de inhoud van het Y-register wordt elke keer met 1 verhoogd.

De RETURN op het eind van de file zorgt er voor dat het programma vervolgt met regel 49447. Nu wordt de waarde \$ BF in de bufferpointer \$ 00A6 geplaatst. De subroutine CHROUT op regel 49451

meent nu dat de buffer vol is en gaat de *gehele* tape I/O buffer, ook het gedeelte dat niet met karakters van de file is gevuld, naar de tape schrijven.

Voor het loaden van een file van de tape moet het secondaire adres \$ 00 zijn. Dit is in overeenstemming met het overeenkomstige commando uit de BASIC programmeertaal.

OPEN 1,1,0

Uiteraard moet het geopende kanaal een input kanaal worden. Ook nu bevat de naam van de file geen enkel nader gegeven. Het volgende programma is voor het loaden van de file van de tape:

LOAD een file van de tape

49408	C100	A9 01	LDA-IMM	01	File nummer.
49410	C102	A2 01	LDX-IMM	01	Device nummer, \$ 01 voor tape.
49412	C104	A0 00	LDY-IMM	00	Secondair adres, \$ 00 voor load file.
49414	C106	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de file.
49417	C109	A9 09	LDA-IMM	09	Lengthe van de naam (SAVE/LOAD).
49419	C10B	A2 00	LDX-IMM	00	Laagste byte tabeladres (naam in tabel).
49421	C10D	A0 CA	LDY-IMM	CA	Hogste byte tabeladres.
49423	C10F	20 BDFF	JSR-ABS	FFBD	SETNAM, invoeren van de naam.
49426	C112	20 C0FF	JSR-ABS	FFC0	OPEN, open de file.
49429	C115	A2 01	LDX-IMM	01	File nummer.
49431	C117	20 C6FF	JSR-ABS	FFC6	CHKIN, kanaal wordt ingaan.
49434	C11A	20 CFFF	JSR-ABS	FFCF	Tape I/O buffer vullen vanaf de tape.
49437	C11D	A9 01	LDA-IMM	01	File nummer.

49439	C11F	20	C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49442	C122	20	E7FF	JSR-ABS	FFE7	CLALL.
49445	C125	A2	00	LDX-IMM	00	Voor de lengte van de file.
49447	C127	E8		INX-IMPL		Verhoog de filelengte.
49448	C128	B0	3C03	LDA-ABS,X	033C	Karakter uit de tape I/O buffer.
49451	C12B	9D	0FCA	STA-ABS,X	CA0F	Karakter in de tabel.
49454	C12E	C9	0D	CMP-IMM	0D	Laatste karakter van de file?
49456	C130	D0	F5	BNE-REL	C127	Voor volgend karakter.
49458	C132	60		RTS-IMPL		RETURN.

Tot en met regel 49431 vertoont het programma niets nieuws. Het werkelijke laden gebeurt in regel 49434. De subroutine CHRIN op \$ FFCF vult dan de tape I/O buffer in zijn totaal met de karakters die van de tape komen. In het eerste gedeelte van de tape I/O buffer bevindt zich de file, afgesloten door een RETURN. Nu de karakters zijn binnengehaald kan het kanaal weer worden gesloten. Hiervoor zijn de regels 49437 tot en met 49442. Nu moeten de karakters worden overgebracht vanuit de tape I/O buffer naar de tabel. Hoewel voor het aanvangsadres van de tape I/O buffer \$ 033C wordt opgegeven bevindt het eerste karakter zich in geheugenplaats \$ 033D. Het index X-register begint met de waarde \$ 01 voor het laden van de accu met de inhoud van register \$ 033D. Deze inhoud wordt weer geschreven in geheugenplaats \$ CA10. Alle karakters uit de buffer worden overgebracht in de tabel. Indien dit gewenst is kan de tabel eventueel nog worden afgesloten met \$ 00. Hiervoor is een kleine aanvulling in het programma nodig.

7.4. Karakters naar de printer

Evenals de disk drive verzorgt de printer zelf al het werk dat nodig is voor het printen van een karakter. Dat houdt in dat de karakters die op het papier moeten worden geprint, stuk voor stuk naar de printer moeten worden verzonden. Deze karakters worden in de printer eerst opgesamd in een buffergeheugen dat in totaal negentig tekens kan bevatten. Als de buffer niet geheel gevuld is met karakters blijft de printer wachten totdat het teken voor RETURN is ontvangen. Pas daarna worden de karakters die zich in de buffer bevinden, geprint. Als de buffer vol is wordt automatisch overgegaan tot printen, om plaats te maken voor nieuwe karakters. Het is steeds noodzakelijk om een serie karakters die moet worden geprint, af te sluiten met een RETURN. Als u de karakters in de computer brengt met de routine GET A\$ dan eindigt u in elk geval met een RETURN. Het volgende programma kunt u voor het printen gebruiken:

File naar de printer

49408	C100	A9	01	LDA-IMM	01	File nummer.
49410	C102	A2	04	LDX-IMM	04	Device nummer, \$ 04 voor de Printer.
49412	C104	A0	00	LDY-IMM	00	Secondair adres, \$ 00 voor upper case.
49414	C106	20	BAFF	JSR-ABS	FFBA	SETLFS, 9e9evens voor de file.
49417	C109	A9	00	LDA-IMM	00	Geen filenaam.
49419	C10B	20	BDFF	JSR-ABS	FFBD	SETNAM.
49422	C10E	20	C0FF	JSR-ABS	FFC0	OPEN, open de file.
49425	C111	A2	01	LDX-IMM	01	File nummer.
49427	C113	20	C9FF	JSR-ABS	FFC9	CHKOUT, kanaal wordt uitgaand.
49430	C116	A0	00	LDY-IMM	00	Lengte van de file.
49432	C118	B9	00CA	LDA-ABS,Y	CA00	Karakter in de ACCU.
49435	C11B	F0	06	BEG-REL	C123	Einde van de file?
49437	C11D	20	D2FF	JSR-ABS	FFD2	Karakter naar de Printer.
49440	C120	C8		INX-IMPL		Verhoog lengte van de file.
49441	C121	D0	F5	BNE-REL	C118	Voor het volgend karakter.
49443	C123	A9	0D	LDA-IMM	0D	Einde file, RETURN.
49445	C125	20	D2FF	JSR-ABS	FFD2	RETURN naar Printer.
49448	C128	A9	01	LDA-IMM	01	File nummer.
49450	C12A	20	C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49453	C12D	20	E7FF	JSR-ABS	FFE7	CLALL.
49456	C130	60		RTS-IMPL		RETURN.

Het heeft geen enkele zin een file voor de printer een naam te geven. Daarom is in regel 49417 voor de lengte van de filenaam \$ 00 ingevoerd. Het adres van de filenaam kan dan uiteraard ook niet worden ingevoerd. Het aanroepen van de subroutine SETNAM is echter wel voorgeschreven. De file heeft een nummer (regel 49408) en het device-nummer voor de printer is \$ 04. Met het secundair adres is iets bijzonders aan de hand. Wordt hiervoor \$ 00 ingevoerd dan staat de printer in de upper case mode (ook wel cursor up mode genoemd) en worden alle karakters in deze mode geprint, tenzij vóór een te printen string (alle karakters tot en met de RETURN) de code voor lower case (CHR\$(17)) wordt verzonden. Wordt echter voor het secundair adres \$ 07 ingevoerd dan worden alle karakters in de lower case mode (cursor down mode) geprint, tenzij vóór een te printen string de code voor upper case mode (CHR\$(145)) wordt verzonden. In dit programma is gekozen voor de upper case mode. Nadat het geopende kanaal tot uitgang is gemaakt worden in de regels 49430 tot en met 49441 de karakters naar de printer verzonden met de subroutine CHROUT. De karakters moeten in de tabel worden ingevoerd, te beginnen met geheugenplaats \$ CA00.

In het programma is een extra RETURN opgenomen (regels 49443 en 49445) zodat in elk geval alle karakters zullen worden geprint.

7.5. Hard Copy van het scherm

Het is mogelijk met de printer een kopie te maken van het scherm, zodat de karakters op dezelfde manier op het papier worden afgedrukt als ze op het scherm staan. Daartoe wordt de inhoud van het schermgeheugen naar de printer gezonden. Het stroomdiagram van het programma geeft figuur 43. Nadat op de gebruikelijke manier het kanaal naar de printer is geopend wordt het schermregeladres (adres van de eerste geheugenplaats van een schermregel) op \$ 0400 gesteld. Nu worden de karakters van deze schermregel naar de printer ge-

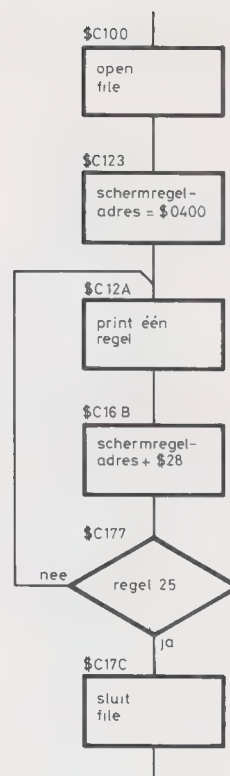


Fig. 43

zonden, inclusief de spaties. De regel wordt afgesloten met een RETURN zodat hij in zijn geheel op het papier wordt gedrukt. Nu wordt het schermregeladres met \$ 28 (40) opgehoogd voor de volgende regel. Zo wordt regel voor regel afgewerkt, tot alle 25 regels van het scherm zijn geprint. Een moeilijkheid daarbij is dat de karakters in het schermgeheugen in schermcode zijn opgeslagen, terwijl de karakters in ASCII code naar de printer moeten worden gestuurd. Een andere moeilijkheid is dat het scherm de karakters zowel in upper case als in lower case kan vertonen, hetgeen uiteraard op het papier op overeenkomstige wijze moet gebeuren. Hoe dat kan worden opgelost laat het volgende programma zien:

HARD COPY

49408 C100	A9 02	LDA-IMM	02	Masker in ACCU.
49410 C102	2D 18D0	AND-ABS	D018	BePaal: upper case of lower case.
49413 C105	F0 04	BEQ-REL	C10B	Spiong bij upper case.
49415 C107	A0 07	LDY-IMM	07	Voor lower case karakters.
49417 C109	D0 02	BNE-REL	C10D	Volgende regel overslaan.
49419 C10B	A0 00	LDY-IMM	00	Voor upper case karakters.
49421 C10D	A9 01	LDA-IMM	01	File nummer.
49423 C10F	A2 04	LDX-IMM	04	Device nummer, \$ 04 voor de Printer.
49425 C111	20 BAFF	JSR-ABS	FFBA	SETLFS, gegevens voor de file.

49428	C114	A9 00	LDA-IMM	00	Geen filenaam.
49430	C116	20 B0FF	JSR-ABS	FFB0	SETNAM.
49433	C119	20 C0FF	JSR-ABS	FFC0	OPEN, open de file.
49436	C11C	A2 01	LDX-IMM	01	File nummer.
49438	C11E	20 C9FF	JSR-ABS	FFC9	CHKOUT, kanaal wordt uitgaand.
49441	C121	A2 00	LDX-IMM	00	Voor het aantal regels.
49443	C123	8A	TXA-IMPL		\$ 00 in ACCU.
49444	C124	85 FB	STA-Z.PAGE		Laagste byte schermregeladres.
49446	C126	A9 04	LDA-IMM	04	\$ 04 in ACCU.
49448	C128	85 FC	STA-Z.PAGE		Hoogste byte schermregeladres.
49450	C12A	A0 00	LDY-IMM	00	Voor het aantal karakters per regel.
49452	C12C	B1 FB	LDA-(IND),Y		Karakter in de ACCU.
49454	C12E	C9 00	CMP-IMM	00	Bepaal inverse of normaal.
49456	C130	30 0E	BMI-REL	C140	SPronk bij normaal karakter.
49458	C132	38	SEC-IMPL		Inverse karakter.
49459	C133	E9 80	SBC-IMM	80	Correctie, (schermcode-128).
49461	C135	48	PHA-IMPL		Bewaar het karakter.
49462	C136	A9 12	LDA-IMM	12	CHR\$(18).
49464	C138	20 D2FF	JSR-ABS	FFD2	CHROUT voor RVS ON.
49467	C13B	A9 01	LDA-IMM	01	
49469	C13D	85 FD	STA-Z.PAGE		Flag voor RVS ON.
49471	C13F	68	PLA-IMPL		Haal karakter terug.
49472	C140	18	CLC-IMPL		Conversie van schermcode naar ASCII.
49473	C141	69 40	ADC-IMM	40	Idem.
49475	C143	C9 A0	CMP-IMM	A0	Idem.
49477	C145	10 0E	BPL-REL	C155	Idem.
49479	C147	69 40	ADC-IMM	40	Idem.
49481	C149	C9 C0	CMP-IMM	C0	Idem.
49483	C14B	10 08	BPL-REL	C155	Idem.
49485	C14D	E9 7F	SBC-IMM	7F	Idem.
49487	C14F	C9 20	CMP-IMM	20	Idem.
49489	C151	10 02	BPL-REL	C155	Idem.
49491	C153	69 40	ADC-IMM	40	Idem.
49493	C155	20 D2FF	JSR-ABS	FFD2	Karakter naar de Printer.
49496	C158	A5 FD	LDA-Z.PAGE		Bepaal RVS flag.
49498	C15A	F0 05	BEO-REL	C161	SPronk bij normaal karakter.
49500	C15C	A9 92	LDA-IMM	92	CHR\$(146).
49502	C15E	20 D2FF	JSR-ABS	FFD2	Voor RVS OFF.
49505	C161	C8	INY-IMPL		Verhoog de karakterteller.
49506	C162	C0 28	CPY-IMM	28	Voor 40 karakters per regel.
49508	C164	D0 C6	BNE-REL	C12C	Voor het volgende karakter.
49510	C166	A9 0D	LDA-IMM	0D	Voor RETURN.
49512	C168	20 D2FF	JSR-ABS	FFD2	Voor het Printen van de regel en NR.
49515	C16B	18	CLC-IMPL		Berekent nieuw schermregeladres.
49516	C16C	98	TYA-IMPL		Idem.
49517	C16D	65 FB	ADC-Z.PAGE		Idem.
49519	C16F	85 FB	STA-Z.PAGE		Idem.
49521	C171	A9 00	LDA-IMM	00	Idem.
49523	C173	65 FC	ADC-Z.PAGE		Idem.
49525	C175	85 FC	STA-Z.PAGE		Idem.
49527	C177	E8	INX-IMPL		Verhoog de regelteller.
49528	C178	E0 19	CPX-IMM	19	Voor 25 regels.
49530	C17A	D0 AE	BNE-REL	C12A	Voor volgende regel.
49532	C17C	A9 01	LDA-IMM	01	File nummer.
49534	C17E	20 C3FF	JSR-ABS	FFC3	CLOSE, sluit het kanaal.
49537	C181	20 E7FF	JSR-ABS	FFE7	CLALL.
49540	C184	60	RTS-IMPL		RETURN.

Of de karakters in de upper case mode of in de lower case mode op het beeldscherm worden geprint hangt samen met de inhoud van geheugenplaats \$ D018. De inhoud hiervan is \$ 15 (00010101) voor

de upper case of \$ 17 (00010111) voor de lower case. Het verschil zit hem in de tweede bit. Met het masker \$ 02 (regel 49408) en een AND instructie (regel 49410) kan de mode worden vastgesteld,

waarmee het secundaire adres \$ 00 voor de upper case of \$ 07 voor lower case kan worden geselecteerd. Het openen van de file volgt hierna. Nu wordt eerst het index X-register als regelteller gereset (\$ 00, regel 49441) waarna het schermregelaadres \$ 0400 in de registers \$ 00FB en \$ 00FC wordt geladen (regels 49443 tot en met 49448). Ook de karakterteller Y wordt gereset (regel 49450). Hierna wordt met de (indirect), Y adresseermethode een karakter opgehaald. Van dit karakter moet eerst worden vastgesteld of het een invers of een normaal karakter is. Bij een invers karakter moet CHR\$ (18) voor RVS ON naar de printer en wordt een flag gezet. In dat geval moet ook de schermcode gecorrigeerd worden (-128). De regels 49472 tot en met 49491 dienen voor de conversie van schermcode naar ASCII. In regel 49493 gaat het karakter in ASCII naar de printer. Steeds als een karakter met RVS ON is geprint wordt RVS weer uitgeschakeld door achter het karakter

CHR\$ (146) naar de printer te sturen (regels 49496 tot en met 49502). Het geheel herhaalt zich voor de volgende karakters tot de gehele regel is afgewerkt (regel 49505 tot en met 49508). Dan wordt \$ 0D voor RETURN naar de printer gestuurd. Dit heeft als resultaat dat de karakters van de regel, die zich nu in de buffer van de printer bevinden, worden geprint en dat de volgende karakters op de volgende regel worden geprint. Nu wordt het schermregelaadres met \$ 28 (40) opgehoogd. Eerst wordt \$ 28 opgeteld bij de lage byte van het adres (\$ 00FB, regels 49515 tot en met 49519). Zou het getal in \$ 00FB hierdoor groter worden dan \$ FF dan wordt het carrybit 1. Door \$ 00 (regel 49521) bij de hoge byte op te tellen met ADC wordt het carrybit bij de hoge byte opgeteld zodat ook de hoge byte van het adres de juiste waarde krijgt. Nu wordt de volgende regel afgewerkt. Dit herhaalt zich tot alle regels zijn behandeld.

8. Bewerkingen met getallen.

8.1. Rekenkundige en logische bewerkingen

Uiteraard is het mogelijk om met machinetaalprogramma's dezelfde bewerkingen uit te voeren die ook in de BASIC programmeertaal mogelijk zijn, mits de betreffende subroutines daarvoor op de juiste manier worden gebruikt.

De rekenkundige bewerkingen zijn steeds bewerkingen tussen twee getallen. Deze getallen zullen in dit hoofdstuk steeds met X en Y aangegeven worden. Het resultaat is Z. De getallen kunnen zowel floating point als integer zijn. Eén van de getallen bevindt zich steeds in FPAC, in de volgende voorbeelden steeds het getal voor Y. Het andere getal (X) bevindt zich in de ruimte voor een variabele, of in een speciaal hulpregister, HLPREG, op

de geheugenplaatsen \$ 0069 tot en met \$ 006E. Na de berekening bevindt zich het resultaat steeds in FPAC.

De getallen kunnen zich in de geheugenruimte bevinden. Een andere mogelijkheid is dat ze worden ingevoerd in FPAC met de INPUT routine.

Ook is het mogelijk dat ze in het programma worden ingevoerd. Zie hiervoor paragraaf 8.4, 'voorbeelden rekenkundige bewerkingen'. In het volgende programma wordt eerst getal X ingevoerd met de INPUT routine en daarna doorgeplaatst in de variabele geheugenruimte (\$ C900).

Dan wordt het getal Y middels de INPUT routine in FPAC geplaatst.

Rekenkundige bewerkingen I

49408 C100	A9 58	LDA-IMM	58	CHR\$(88), voor "X".
49410 C102	20 D2FF	JSR-ABS	FFD2	Print "X".
49413 C105	20 3DC1	JSR-ABS	C13D	Subroutine input.
49416 C108	A2 00	LDX-IMM	00	Laag byte van adres voor variabele X.
49418 C10A	A0 C9	LDY-IMM	C9	Hog byte van adres voor variabele X.
49420 C10C	20 D4BB	JSR-ABS	BBD4	Waarde voor X in geheugenplaats.
49423 C10F	A9 59	LDA-IMM	59	CHR\$(89) voor "Y".
49425 C111	20 D2FF	JSR-ABS	FFD2	Print "Y".
49428 C114	20 3DC1	JSR-ABS	C13D	Subroutine input.
49431 C117	A9 00	LDA-IMM	00	Laag byte variabele X.
49433 C119	A0 C9	LDY-IMM	C9	Hog byte variabele X.
49435 C11B	20 67B8	JSR-ABS	B867	Optellen van FPAC (Y) en X.
49438 C11E	A2 05	LDX-IMM	05	Laag byte somadres.
49440 C120	A0 C9	LDY-IMM	C9	Hog byte somadres.
49442 C122	20 D4BB	JSR-ABS	BBD4	Som in geheugenruimte.
49445 C125	20 DBBD	JSR-ABS	BDD4	Van FLOATING POINT naar ASCII.
49448 C128	A2 00	LDX-IMM	00	Voor de stringlengte.
49450 C12A	BD 0001	LDA-ABS,X	0100	Haal karakter op.
49453 C12D	F0 06	BEQ-REL	C135	Einde string?
49455 C12F	20 D2FF	JSR-ABS	FFD2	Print karakter.
49458 C132	E8	INX-IMPL		Verhoog stringlengte.
49459 C133	D0 F5	BNE-REL	C12A	Voor volgend karakter.
49461 C135	A9 0D	LDA-IMM	0D	Einde string, voor RETURN.
49463 C137	20 D2FF	JSR-ABS	FFD2	Cursor naar nieuwe regel.
49466 C13A	4C 00C1	JMP-ABS	C100	Voor volgende berekening.
49469 C13D	A9 3F	LDA-IMM	3F	Subroutine input.
49471 C13F	20 D2FF	JSR-ABS	FFD2	
49474 C142	A2 FF	LDX-IMM	FF	
49476 C144	E8	INX-IMPL		
49477 C145	20 CFFF	JSR-ABS	FFCF	
49480 C148	9D 00C8	STA-ABS,X	C800	
49483 C14B	C9 0D	CMP-IMM	0D	
49485 C14D	D0 F5	BNE-REL	C144	
49487 C14F	20 D2FF	JSR-ABS	FFD2	
49490 C152	A9 00	LDA-IMM	00	

```

49492 C154 85 22 STA-Z.PAGE
49494 C156 A9 C8 LDA-IMM      C8
49496 C158 85 23 STA-Z.PAGE
49498 C15A 8A   TXA-IMPL
49499 C15B 20 B5B7 JSR-ABS      B7B5
49502 C15E 60   RTS-IMPL

```

Het invoeren van de getallen gebeurt zoals hier-voor beschreven in de regels 49408 tot en met 49428. De rekenkundige bewerking heeft hier plaats tussen X in de geheugenruimte en Y in FPAC. Het adres van X wordt daartoe in het X-register (lage byte) en in het Y-register (hoge byte) geladen. Subroutine ADD op regel 494435 voert de bewerking 'optellen' uit. Het resultaat (in FPAC) wordt in de geheugenruimte geplaatst en ook op de gebruikelijke wijze geprint. Start het programma met SYS 49408. Na het invoeren van de waarden voor X en Y ziet u het resultaat op het beeldscherm geprint.

Het programma wordt ook voor vermenigvuldigen gebruikt. Hiervoor moet dan in regel 49435 de subroutine MULTIPLY worden aangeroepen. Lijst 7 geeft de subroutines die bij dit programma kunnen worden gebruikt.

Lijst 7. Rekenkundige bewerkingen

Bewerking	Subroutine	adres
Z = X + Y	ADD	B867
Z = X * Y	MULTIPLY	BA28

Voor de andere bewerkingen moet het volgende programma worden gebruikt:

Rekenkundige bewerkingen II

```

49408 C100 A9 58 LDA-IMM      58  CHR$(88) voor "X".
49410 C102 20 D2FF JSR-ABS      FFD2  Print "X".
49413 C105 20 40C1 JSR-ABS      C140  Subroutine inPut.
49416 C108 A2 00 LDX-IMM      00  Lage byte voor X.
49418 C10A A0 C9 LDY-IMM      C9  Hoge byte voor X.
49420 C10C 20 D4BB JSR-ABS      BBD4  Waarde voor X in geheugenplaats.
49423 C10F A9 59 LDA-IMM      59  CHR$(89) voor "Y".
49425 C111 20 D2FF JSR-ABS      FFD2  Print "Y".
49428 C114 20 40C1 JSR-ABS      C140  Subroutine inPut.
49431 C117 A9 00 LDA-IMM      00  Lage byte variabele X.
49433 C119 A0 C9 LDY-IMM      C9  Hoge byte variabele X.
49435 C11B 20 8CBA JSR-ABS      BA8C  X in hulpregister HLPREG.
49438 C11E 20 53B8 JSR-ABS      B853  Aftrekken, X-Y, HLPREG-FPAC naar FPAC.
49441 C121 A2 05 LDX-IMM      05  Lage byte verschiladres.
49443 C123 A0 C9 LDY-IMM      C9  Hogebyte verschiladres.
49445 C125 20 D4BB JSR-ABS      BBD4  Verschil in geheugenruimte.
49448 C128 20 DDBD JSR-ABS      BDDD  Van FLOATING POINT naar ASCII.
49451 C12B A2 00 LDX-IMM      00  Voor de stringlengte.
49453 C12D BD 0001 LDA-ABS,X    0100  Haal karakter op.
49456 C130 F0 06 BEQ-REL      C138  Einde string?
49458 C132 20 D2FF JSR-ABS      FFD2  Print karakter.
49461 C135 E8   INX-IMPL
49462 C136 D0 F5 BNE-REL      C12D  Verhoog stringlengte.
49464 C138 A9 0D LDA-IMM      0D  Voor volgend karakter.
49466 C13A 20 D2FF JSR-ABS      FFD2  Einde string, voor RETURN.
49469 C13D 4C 00C1 JMP-ABS      C100  Cursor naar nieuwe regel.
49472 C140 A9 3F LDA-IMM      3F  Voor volgende berekening.
49474 C142 20 D2FF JSR-ABS      FFD2  Subroutine inPut.
49477 C145 A2 FF LDX-IMM      FF
49479 C147 E8   INX-IMPL
49480 C148 20 CFFF JSR-ABS      FFCF
49483 C14B 9D 00C8 STA-ABS,X    C800
49486 C14E C9 0D CMP-IMM      0D
49488 C150 D0 F5 BNE-REL      C147
49490 C152 20 D2FF JSR-ABS      FFD2
49493 C155 A9 00 LDA-IMM      00
49495 C157 85 22 STA-Z.PAGE

```



```

49497 C159 A9 C8 LDA-IMM C8
49499 C15B 85 23 STA-Z.PAGE
49501 C15D 8A TXA-IMPL
49502 C15E 20 B5B7 JSR-ABS B7B5
49505, C161 60 RTS-IMPL

```

Het enige verschil met het vorige programma is dat op regel 49435 de waarde voor X met de subroutine variabele in HLPREG in het hulpregister wordt geplaatst. De bewerking is nu tussen FPAC en HLPREG.

De bewerking die wordt uitgevoerd is afhankelijk van de subroutine die in regel 49438 wordt aangeroepen. Aftrekken, delen, machtsverheffen en de logische bewerkingen AND en OR kunnen door dit programma worden uitgevoerd. Lijst 8 geeft de subroutines. Let hierbij op de volgorde van de variabelen X en Y!

U hoeft achter de opcode voor JSR, \$ 20, in regel 49438 slechts het adres van de subroutine in te voeren die de bewerking uitvoert die u wenst.

Lijst 8. Rekenkundige en logische bewerkingen

Bewerking	Subroutine	adres
Z=X-Y	SUBTRACT	B853
Z=X/Y	DIVIDE	BBOC
Z=X↑Y	POWER	BF7B
Z=XANDY	AND	AFE9
Z=XORY	OR	AFE6

8.2. Functies

Voor het laten uitvoeren van een functie, zoals SIN(X) in $Z=Y*\text{SIN}(X)$, heeft slechts één getal te worden ingevoerd. Een routine die een functie uitvoert verwacht het getal in FPAC en zal het resultaat van de functie ook weer in FPAC plaatsen. Het volgende programma kan voor een functie worden gebruikt:

Functies I

```

49408 C100 A9 58 LDA-IMM 58 CHR$(88), voor "X".
49410 C102 20 D2FF JSR-ABS FFD2 Print "X".
49413 C105 20 2AC1 JSR-ABS C12A Subroutine input.
49416 C108 20 71BF JSR-ABS BF71 Subroutine SQR, SQR(X) in FPAC.
49419 C10B A2 05 LDX-IMM 05 Laagste byte resultaatadres.
49421 C10D A0 C9 LDY-IMM C9 Hoge byte resultaatadres.
49423 C10F 20 D4BB JSR-ABS BBD4 Resultaat in geheugenruimte.
49426 C112 20 DDDB JSR-ABS BDDD Van FLOATING POINT naar ASCII.
49429 C115 A2 00 LDX-IMM 00 Voor de stringlengte.
49431 C117 BD 0001 LDA-ABS,X 0100 Haal karakter op.
49434 C11A F0 06 BEQ-REL C122 Einde string?
49436 C11C 20 D2FF JSR-ABS FFD2 Print karakter.
49439 C11F E8 INX-IMPL Verhoog stringlengte.
49440 C120 D0 F5 BNE-REL C117 Voor volgend karakter.
49442 C122 A9 0D LDA-IMM 0D Einde string, voor RETURN.
49444 C124 20 D2FF JSR-ABS FFD2 Cursor naar nieuwe regel.
49447 C127 40 00C1 JMP-ABS C100 Voor volgende berekening.
49450 C12A A9 3F LDA-IMM 3F Subroutine input.
49452 C12C 20 D2FF JSR-ABS FFD2
49455 C12F A2 FF LDX-IMM FF
49457 C131 E8 INX-IMPL
49458 C132 20 CFFF JSR-ABS FFCF
49461 C135 9D 00C8 STA-ABS,X C800
49464 C138 C9 0D CMP-IMM 0D
49466 C13A D0 F5 BNE-REL C131
49468 C13C 20 D2FF JSR-ABS FFD2
49471 C13F A9 00 LDA-IMM 00
49473 C141 85 22 STA-Z.PAGE
49475 C143 A9 C8 LDA-IMM C8
49477 C145 85 23 STA-Z.PAGE
49479 C147 8A TXA-IMPL
49482 C148 20 B5B7 JSR-ABS B7B5
49483 C14B 60 RTS-IMPL

```

In de eerste drie regels wordt het getal voor X ingevoerd in FPAC. In regel 49416 wordt de betreffende subroutine aangeroepen, in dit geval de subroutine voor $Y = \text{SQR}(X)$. In plaats van deze subroutine kunt u het adres van elke andere subroutine die wordt vermeld in lijst 9, in deze regel invoeren.

8.3. Het toevalsgetal en pi

Een functie waarbij de grootte van het getal in FPAC vóór het aanroepen van deze functie er niets toe doet, zolang het maar positief is, is de $\text{RND}(X)$ functie. Het volgende programma zal u dan ook, steeds als een willekeurige toets wordt ingedrukt, een randomgetal op het beeldscherm printen.

Lijst 9. Functies

Bewerking	Subroutine	Adres
$Y = \text{SQR}(X)$	SQR	BF71
$Y = \text{SIN}(X)$	SIN	E26B
$Y = \text{COS}(X)$	COS	E264
$Y = \text{TAN}(X)$	TAN	E2B4
$Y = \text{ATN}(X)$	ATN	E30E
$Y = \text{LOG}(X)$	LOG	B9EA
$E = e^X$	EXP	BFED
$Y = \text{INT}(X)$	INT	BCCC
$Y = \text{SGN}(X)$	SGN	BC39
$Y = \text{ABS}(X)$	ABS	BC58
$Y = -X$	NEG	BFB4

Functies II

```

49408 C100 A5 C5 LDA-Z.PAGE
49410 C102 C9 40 CMP-IMM
49412 C104 D0 FA BNE-REL
49414 C106 A5 C5 LDA-Z.PAGE
49416 C108 C9 40 CMP-IMM
49418 C10A F0 FA BEQ-REL
49420 C10C 20 97E0 JSR-ABS
49422 C10E A2 00 LDX-IMM
49424 C110 A0 C9 LDY-IMM
49426 C112 20 D4BB JSR-ABS
49428 C114 20 DDD0 JSR-ABS
49430 C116 A2 00 LDX-IMM
49432 C118 E0 0001 LDA-ABS,X
49434 C11A F0 06 BEQ-REL
49436 C11C 20 D4FF JSR-ABS
49438 C11E E0 INY-IMPL
49440 C120 D0 F5 BNE-REL
49442 C122 A0 00 LDA-IMM
49444 C124 20 D4FF JSR-ABS
49446 C126 40 0001 JMP-ABS

```

```

40 BePaal de inhoud van re9. 197.
C100 Is de toets no9 ingedrukt?
40 Wacht tot hij wordt losgelaten.
C106 BePaal de inhoud van re9. 197.
E097 Ga na of een toets is ingedrukt.
00 Wacht op een toets.
00 Subroutine voor  $X = \text{RND}(0)$ .
C9 La9e byte variabele X.
BBD4 Hoge byte variabele X.
BDD0 X in geheugenruimte.
00 Van FLOATING POINT naar ASCII.
0100 Voor de stringlengte.
C126 Haal karakter op.
FFD2 Einde string?
C108 Print karakter.
C11B Verhoog stringlengte.
00 Voor volgend karakter.
FFD2 Einde string, voor RETURN.
C100 Cursor naar de nieuwe regel.

```

Voor dit programma zijn de routines 'wacht op toets los' (regels 49408 tot en met 49412) en 'wacht op toets in' (regels 49414 tot en met 49418) gebruikt. Daarna wordt de RND subroutine op \$E097 aangeroepen. Deze subroutine genereert een random getal en plaatst dit in FPAC. Het verdere programma verloopt overeenkomstig aan het voorgaande. Het getal dat de subroutine genereert is alleen van het getal X in FPAC afhankelijk als dit een negatief getal is. In dat geval is het gegenereerde getal *geen* random getal. Indien nodig moet daarom FPAC voor het aanroepen van de subrou-

tine eerst met een positief getal worden gevuld. Bij dit programma is, nadat de eerste keer op een toets wordt gedrukt, het getal in FPAC positief (het randomgetal is altijd positief).

Het getal π kunt u nog wel eens nodig hebben omdat de waarde van hoeken bij goniometrische functies steeds in radialen moet worden ingevoerd. Het getal voor π is in de computer opgeslagen in de geheugenplaatsen \$AEA8 tot en met \$AEAC. Het volgende programma plaatst het getal in de geheugenruimte voor een variabele en print het op het beeldscherm:

π

```

49408 C100 A2 04 LDX-IMM
49410 C102 BD A8AE LDA-ABS,X
49412 C104 9D 00C9 STA-ABS,X

```

```

04 Voor 5 geheugenplaatsen.
AEA8 Getal uit constante-adres.
C900 Getal in variabele-adres.

```

49416	C108	CA	DEX-IMPL		X=X-1.
49417	C109	10 F7	BPL-REL	C102	Voor volgende geheugenplaats.
49419	C10B	A9 00	LDA-IMM	00	La9e byte variabele-adres.
49421	C10D	A0 C9	LDY-IMM	C9	Hoge byte variabele-adres.
49423	C10F	20 A2BB	JSR-ABS	BBA2	Getal voor P1 in FPAC.
49426	C112	20 DDBD	JSR-ABS	BDDD	Van FLOATING POINT naar ASCII.
49429	C115	A2 00	LDX-IMM	00	Voor de stringlengte.
49431	C117	BD 0001	LDA-ABS,X	0100	Haal het karakter op.
49434	C11A	F0 06	BEQ-REL	C122	Einde string?
49436	C11C	20 D2FF	JSR-ABS	FFD2	Print het karakter.
49439	C11F	E8	INX-IMPL		Verhoog de stringlengte.
49440	C120	D0 F5	BNE-REL	C117	Voor volgend karakter.
49442	C122	A9 0D	LDA-IMM	0D	Einde string, voor RETURN.
49444	C124	20 D2FF	JSR-ABS	FFD2	Cursor naar een nieuwe regel.
49447	C127	60	RTS-IMPL		RETURN.

Omdat het index-X register de waarde \$ 04 heeft wordt eerst de inhoud van geheugenplaats \$ AEA8 + \$ 04 = \$ AEAC in de accu geladen. Deze plaatst deze waarde in geheugenplaats \$ C900 + \$ 04 = \$ C904. Nu wordt het index-X register met 1 verminderd en herhaalt de werking zich voor de volgende lagere geheugenplaatsen. Als X=0 dan is het getal voor π overgebracht en volgt het schrijven naar het scherm. Hiertoe moet het getal π eerst in FPAC worden geplaatst (regels 49419 tot en met 49423). De rest van

het programma verloopt volgens de bekende weg voor het printen van een getal.

8.4. Voorbeelden van rekenkundige bewerkingen

In deze paragraaf vindt u twee voorbeelden van rekenkundige bewerkingen, waarbij de benodigde gegevens niet met een INPUT routine in de computer worden gebracht maar ingevoerd zijn in het programma. Als eerste voorbeeld de berekening

$$X = 475 * \sin(\pi/6)$$

*PRINT 475*SIN ($\pi/6$)*

49408	C100	A9 36	LDA-IMM	36	# 54
49410	C102	8D 00C8	STA-ABS	C800	"6" in de buffer.
49413	C105	A9 01	LDA-IMM	01	Voor 1 karakter.
49415	C107	20 4EC1	JSR-ABS	C14E	Van ASCII naar FPAC, getal 6 in FPAC.
49418	C10A	A9 A8	LDA-IMM	A8	La9e byte van adres constante Pi.
49420	C10C	A0 AE	LDY-IMM	AE	Hoge byte van adres constante Pi.
49422	C10E	20 8CBA	JSR-ABS	BA8C	Pi in HLPREG.
49425	C111	20 0CBB	JSR-ABS	BB0C	DIVIDE, Pi/6 in FPAC.
49428	C114	20 6BE2	JSR-ABS	E26B	SIN, SIN(Pi/6) in FPAC.
49431	C117	A2 00	LDX-IMM	00	La9e byte variabele-adres.
49433	C119	A0 C9	LDY-IMM	C9	Hoge byte variabele-adres.
49435	C11B	20 D4BB	JSR-ABS	BBD4	SIN(Pi/6) in geheugenruimte.
49438	C11E	A9 34	LDA-IMM	34	# 52
49440	C120	8D 00C8	STA-ABS	C800	"4" in de buffer.
49443	C123	A9 37	LDA-IMM	37	# 55
49445	C125	8D 01C8	STA-ABS	C801	"7" in de buffer.
49448	C128	A9 35	LDA-IMM	35	# 53
49450	C12A	8D 02C8	STA-ABS	C802	"5" in de buffer.
49453	C12D	A9 03	LDA-IMM	03	Voor 3 karakters.
49455	C12F	20 4EC1	JSR-ABS	C14E	Van ASCII naar FPAC, getal 475 in FPAC.
49458	C132	A9 00	LDA-IMM	00	La9e byte variabele-adres.
49460	C134	A0 C9	LDY-IMM	C9	Hoge byte variabele-adres.
49462	C136	20 28BA	JSR-ABS	BA28	MULTIPLY, 475*sin(Pi/6) in FPAC.
49465	C139	20 DDBD	JSR-ABS	BDDD	Conversie van FPAC in ASCII.
49468	C13C	A2 00	LDX-IMM	00	Voor de stringlengte.
49470	C13E	BD 0001	LDA-ABS,X	0100	Haal karakter op.
49473	C141	F0 06	BEQ-REL	C149	Einde string?
49475	C143	20 D2FF	JSR-ABS	FFD2	Print karakter.
49478	C146	E8	INX-IMPL		Verhoog de stringlengte.
49479	C147	D0 F5	BNE-REL	C13E	Voor volgend karakter.

49481	C149	A9 0D	LDA-IMM	0D	Einde string, voor RETURN.
49483	C14B	4C D2FF	JMP-ABS	FFD2	Cursor naar nieuwe regel, RETURN.
49486	C14E	A2 00	LDX-IMM	00	SUBROUTINE, lage byte bufferadres.
49488	C150	84 22	STX-Z.PAGE		Pointer laag.
49490	C152	A2 08	LDX-IMM	08	Hoge byte bufferadres.
49492	C154	86 23	STX-Z.PAGE		Pointer hoog.
49494	C156	20 B5E7	JSR-ABS	B7B5	Van ASCII naar FLOATING POINT in FPAC.
49497	C159	80	RTS-IMPL		RETURN.

Deze opgave wordt op dezelfde manier aangepakt als met een gewone calculator: eerst $\pi/6$ berekenen, daarna $\text{SIN}(\pi/6)$ bepalen en uiteindelijk de vermenigvuldiging $475 * \text{SIN}(\pi/6)$ uitvoeren. Voor een deling moet het deeltal (π) in het hulpregister HLPREG en de deler (6) in FPAC. Eerst wordt het getal 6 in FPAC geplaatst. Daartoe moet het karakter '6' in de buffer worden geladen waarna het met de subroutine ASCII naar floating point in FPAC kan worden geplaatst. Dit laatste is in dit programma als een subroutine uitgevoerd omdat de gelijke werking ook nog voor het getal 475 nodig is (regels 49486 en verder). Eerst wordt de stringlengte (\$ 01) in de accu geplaatst (regel 49413) waarna de subroutine het verdere werk doet.

In de regels 49418 tot en met 49422 wordt het getal π in HLPREG geplaatst. Dit kan door direct het adres van deze constante in de geheugenruimte (\$AEA8) in te voeren en de subroutine floating point — HLPREG AAN TE ROEPEN. Nu kan de

subroutine DIVIDE zijn werk doen. Omdat het resultaat hiervan zich in FPAC bevindt kan direct de subroutine SIN worden gebruikt. Het resultaat hiervan wordt naar de geheugenruimte voor een variabele overgebracht. De vermenigvuldiging die nu nog tot stand moet komen wordt uitgevoerd tussen een getal in FPAC (475) en één in de geheugenruimte ($\text{SIN}(\pi/6)$). Daarom wordt het getal 475 ingevoerd in de buffer (karakter voor karakter) en van ASCII naar floating point omgezet (regels 49438 tot en met 49455).

Nu wordt de vermenigvuldiging uitgevoerd (regels 49458 tot en met 49462) zodat het resultaat op de gebruikelijke wijze kan worden geprint. Regel 49483 is de laatste regel van het hoofdprogramma. Hier ziet u een JMP naar \$ FFD2. Dit is de subroutine CHROUT waarvan we de RTS gebruiken om het programma te verlaten.

Het tweede voorbeeld is voor de berekening

$$X = \text{INT}(6 * \text{RND}(1)) + 1$$

*PRINT INT (6*RND(1)) + 1*

49403	C100	A9 35	LDA-IMM	36	# 54
49410	C102	20 41C1	JSR-ABS	C141	Getal 6 in FPAC.
49413	C105	A2 00	LDX-IMM	00	Lage byte variabele-adres.
49415	C107	A0 09	LDY-IMM	09	Hoge byte variabele-adres.
49417	C109	20 D4BE	JSR-ABS	BBD4	"6" in geheugenruimte.
49420	C10C	A9 31	LDA-IMM	31	# 49
49422	C10E	20 41C1	JSR-ABS	C141	Getal 1 in FPAC.
49425	C111	A2 05	LDX-IMM	05	Lage byte variabele-adres.
49427	C113	A0 09	LDY-IMM	09	Hoge byte variabele-adres.
49429	C115	20 D4BD	JSR-ABS	BBD4	"1" in geheugenruimte.
49432	C118	20 B7E0	JSR-ABS	E097	RND, RND(1) in FPAC.
49435	C11B	A9 00	LDA-IMM	00	Lage byte adres getal "6".
49437	C11D	A0 09	LDY-IMM	09	Hoge byte adres getal "6".
49439	C11F	20 23BA	JSR-ABS	B928	MULTIPLY, 6*RND(1) in FPAC.
49442	C122	20 C02C	JSR-ABS	B0CC	INT, INT(6*RND(1)) in FPAC.
49445	C125	A9 05	LDA-IMM	05	Lage byte adres getal "1".
49447	C127	A0 09	LDY-IMM	09	Hoge byte adres getal "1".
49449	C129	20 B7B3	JSR-ABS	B867	ADD, INT(6*RND(1))+1 in FPAC.
49452	C12C	20 DDBD	JSR-ABS	BDD0	Conversie van FPAC in ASCII.
49455	C12F	A2 00	LDX-IMM	00	Voor de stringlengte.
49457	C131	BD 0001	LDA-ABS,X	0100	Haal karakter op.
49460	C134	F0 06	BEQ-REL	C13C	Einde string?
49462	C136	20 D2FF	JSR-ABS	FFD2	Print karakter.
49465	C139	E9	INCL-IMPL		Verhoog de stringlengte.

49495 C13A	D8 F0	BNE-BLL	C131	Voor volgend karakter.
49496 C13C	89 0D	LDA-INN	0D	Einde string, voor RETURN.
49497 C13E	4C D2FF	JMP-RAL	FFD2	Cursor naar nieuwe regel, RETURN.
49498 C141	3D 03C6	STA-RBS	C609	SUBROUTINE. Karakter in de buffer.
49499 C143	89 01	LDA-INN	01	Voor 1 karakter.
49500 C145	8C 06	LDA-INN	06	Laag byte bufferadres.
49501 C147	3E 12	STL-Z.PAGE		Pointer laag.
49502 C149	8C 07	LDA-INN	07	Hoge byte bufferadres.
49494 C14C	3E 37	STL-Z.PAGE		Pointer hoog.
49495 C14E	23 B2E7	STL-RBS	E7B5	Van ASCII naar FLOATING POINT in FPAC.
49496 C150	10	RTS-INPL		RETURN.

Het programma begint met het invoeren van de getallen 6 en 1. Het getal 1 het laatst. Beide getallen worden in de geheugenruimte voor variabelen geplaatst, 6 te beginnen op C900 en 1 te beginnen op \$ C905.

In regel 49432 is het getal 1 ook nog in FPAC, zodat door het aanroepen van de subroutine RND inderdaad RND(1) in FPAC wordt geplaatst.

Zoals reeds is vermeld geeft elk positief getal in FPAC de juiste werking van de subroutine RND. Voor de vermenigvuldiging van RND(1) met 6 wordt het adres van het getal 6 ingevoerd en de subroutine MULTIPLY aangeroepen. De daarop volgende subroutine INT zorgt voor INT(6*RND(1)). De optelling met het getal 1 geschiedt op dezelfde wijze als de vermenigvuldiging: de bewerking vindt plaats tussen het getal in FPAC en een getal in de geheugenruimte. Na deze bewerking wordt het resultaat geprint.

8.5. De USER functie

Tot nu toe is steeds gesteld dat een machinetaalprogramma vanuit BASIC moet worden gestart met SYS(SA), waarin SA het startadres van het machinetaalprogramma is. Er is nog een ander statement waarmee vanuit BASIC een machine programma kan worden aangeroepen. Dit is het `USR(X)` statement. In het statement is X een getal waarmee het machinetaalprogramma een bewerking moet uitvoeren. Bij het uitvoeren van de instructie `USR(X)` wordt het getal X in FPAC geplaatst. Op welk adres het machinetaalprogramma moet starten hangt af van de vector in de adressen \$ 0311 en \$ 0312. De handelswijze voor het toepassen van de user functie is als volgt:

- Het startadres van het machinetaalprogramma in \$ 0311 en \$ 0312 laden. Het machinetaalprogramma moet afgesloten zijn met RTS.
- Op de juiste plaats in het BASIC programma het statement `USR(X)` opnemen.

Als voorbeeld nemen we de berekening $Y = 475 * \sin(\pi/6)$ uit paragraaf 8.4. Voor $\sin(\pi/6)$ gebruiken we de subroutine SIN op \$ E26B, lage byte \$ 6B, 107 decimaal en hoge byte \$ E2, 226 decimaal. Het (BASIC) programma verloopt nu als volgt:

USER functie

```
10 POKE 785,107:POKE 786,226
20 X=PI/6
30 Y=USR(X)
40 Z=475*Y:PRINT Z
```

In regel 10 wordt het startadres van het machinetaalprogramma voor de sin functie ingevoerd, $785 \triangleq \$ 0311$, $786 \triangleq \$ 0312$. De variabele X wordt berekend in regel 20 terwijl de subroutine SIN in regel 30 in werking wordt gesteld. Het resultaat hiervan vinden we uiteindelijk weer terug in de variabele Y van het BASIC programma zodat in regel 40 aan Z de juiste waarde kan worden gegeven.

8.6. Het vergelijken van twee getallen

Het vergelijken van twee variabelen X en Y is mogelijk door de twee van elkaar af te trekken. Hiervoor kan de routine 'rekenkundige bewerkingen II' worden gebruikt. De geheugenplaatsen \$ 0061 en \$ 0066 geven dan uitslag over de verhouding van de variabelen onderling. De mogelijkheden zijn:

$X \geq Y$; $X = Y$ en $X < Y$.

Voor het controleren of twee variabelen gelijk zijn gebruiken we geheugenplaats \$ 0061. Eerst de twee variabelen van elkaar aftrekken en dan bijvoorbeeld:

```
LDA Z,PAGE $ 0061
BNE $ (sprongadres)
```

Om na te gaan of één van de getallen groter is dan de ander zijn na de aftrekking de volgende mogelijkheden:

LDA Z.PAGE \$ 0066

BPL \$ (sprongadres)

Sprong als X gelijk aan of groter is dan Y: $X \geq Y$.

Geen sprong als X kleiner is dan Y: $X < Y$

LDA Z.PAGE \$ 0066

BMI \$ (sprongadres)

Sprong als X kleiner is dan Y: $X < Y$.

Geen sprong als X groter is dan of gelijk is aan Y:
 $X \geq Y$.

9. Grafische mogelijkheden

9.1. Het printen

Alle printhandelingen op het beeldscherm kunnen worden uitgevoerd met de subroutine CHROUT op \$ FFD2. Dat geldt niet alleen voor het printen van de karakters maar ook voor het kiezen van de kleur van de tekst, het 'schoonmaken' van het scherm (CLR HOME), CRSR HOME en de vier cursorbewegingen. Elke handeling komt tot stand door het aanroepen van de subroutine CHROUT, vooraf gegaan door het vullen van de accu (LDA) met de ASCII code die voor de betreffende handeling is gegeven in appendix F van de handleiding. Deze codes komen overigens niet altijd overeen

met de officiële ASCII code (commodore ASCII). Er zijn nog een aantal subroutines die ons het printen kunnen vergemakkelijken, zoals het printen van een string. Zie hiervoor ook paragraaf 6.4. Ook de *plaats* op het scherm waar het printen zal aanvangen (de plaats van de cursor) is eenvoudig te bepalen. Hiervoor kunnen we de subroutine PLOT op \$ FFF0 gebruiken. Met deze subroutine kunnen we niet alleen de cursor op een bepaalde plaats op het scherm brengen maar we kunnen ook hiermee bepalen op welke plaats zich de cursor bevindt. Probeer eerst eens het volgende programma:

Bepaal de plaats van de cursor

49408 C100	38	SEC-IMPL	Carry bit 1.
49409 C101	20 F0FF	JSR-ABS	FFF0 PLOT, bepaal de Plaats van de cursor.
49412 C104	84 FB	STY-Z.PAGE	Kolomnummer in # 251.
49414 C106	86 FC	STX-Z.PAGE	Regelnummer in 252.
49416 C108	60	RTS-IMPL	RETURN.

Bepalend voor de werking van de subroutine PLOT is de toestand van het carry bit. Dit is hier '1' gemaakt. De subroutine bepaalt dan de plaats waar de cursor is en geeft het X-register de waarde van het regelnummer en het Y-register de waarde van het kolomnummer.

Zoals u weet is het scherm verdeeld in 25 (horizontale) regels die elk 40 karakters kunnen bevatten. Deze veertig karakters per regel vormen 40 (verticale) kolommen die van 0 tot en met 39 zijn genummerd. De 25 regels zijn van 0 tot en met 24 genummerd.

Start het programma met

SYS49408:PRINT PEEK(251),PEEK(252)

Door het RETURN waarmee de bovenstaande regel wordt uitgevoerd gaat de cursor eerst naar het begin van de volgende regel. U zult dus voor het kolomnummer een '0' op het scherm geprint zien. Voer, vóórdat u het volgende programma probeert, eerst door middel van de GET AS routine de volgende tekst in de tabel:

MICROCOMPUTER.

Nu kunt u het volgende programma in de computer brengen:

Besturing van de cursor

49408 C100	20 44E5	JSR-ABS	E544	CLEAR SCREEN, gelijk aan CLR HOME.
49411 C103	A2 0C	LDX-IMM	0C	# 12, cursor naar regel 12.
49413 C105	A0 0A	LDY-IMM	0A	# 10, cursor naar kolom 10.
49415 C107	18	CLC-IMPL		Carry bit 0.
49416 C108	20 F0FF	JSR-ABS	FFF0	Voor het Plaatsen van de cursor.
49419 C10B	A3 00	LDA-IMM	00	Laag byte tabeladres.
49421 C10D	A0 0A	LDY-IMM	0A	Hoog byte tabeladres.
49423 C10F	20 1EAB	JSR-ABS	AB1E	Print de string.
49426 C112	4C 03C1	JMP-ABS	C103	Herhaal.

De subroutine PLOT in regel 49416 zal nu de cursor naar regelnummer 12 (X-register) en kolomnummer 10 (Y-register) sturen. In dit programma is daarvoor het carry bit '0' gemaakt. Het regelnummer is altijd een getal van 0 tot en met 24 en het kolomnummer een getal van 0 tot en met 39. Vooraf is in regel 49408 een nieuwe subroutine gebruikt.

CLEAR SCREEN op \$ E544. Deze subroutine heeft dezelfde werking als print CHR\$(147), voor CLR HOME. Dat wil zeggen dat het gehele scherm wordt 'gewist' en dat de cursor naar de eerste plaats van het scherm gaat. Dit laatste heeft hier geen waarde omdat hij door subroutine PLOT direct weer naar een andere plaats wordt gedirigeerd. Er is overigens ook nog een subroutine HOME op \$ E566. Door deze subroutine wordt het scherm niet gewist maar gaat de cursor alleen naar de eerste plaats op het scherm. De regels 49419 tot en met 49423 printen de zin die we voor-

af hebben ingevoerd op de plaats die de cursor nu aangeeft en komt daarmee netjes op het midden van het scherm te staan.

Als laatste had een RTS kunnen volgen. Dan had echter het woord READY het scherm ontsierd. Vandaar dat is terug gegaan naar regel 49411 voor het opnieuw printen van de tekst, op dezelfde plaats. Probeer eens wat voor effect het heeft als u het programma terug laat gaan naar regel 49408 (JMP \$ C100)!

9.2. Bewegende beelden

Het geheim van de bewegende beelden kent u al van het BASIC programmeren: de afbeelding (wat dat dan ook mag zijn) moet op de plaats direct naast zijn huidige plaats wordt geprint, waarbij de huidige plaats moet worden gewist door daar een spatie in te schrijven. Bij het volgende programma wordt als afbeelding de gevulde cirkel gebruikt (CHR\$(113)) bij wijze van bal, die we op een horizontale regel over het scherm laten verplaatsen.

Bewegende beelden

49408 C100	A9 05	LDA-IMM	05	# 5, voor CHR\$(5).
49410 C102	20 D2FF	JSR-ABS	FFD2	Voor witte tekens.
49412 C105	20 44E5	JSR-ABS	E544	CLEAR SCREEN, voor CLR HOME.
49414 C108	A2 02	LDX-IMM	02	Regelnummer.
49416 C10A	A0 00	LDY-IMM	00	Cursor aan het begin van de regel.
49420 C10C	A9 71	LDA-IMM	71	# 113, voor CHR\$(113) (bal).
49422 C10E	20 28C1	JSR-ABS	C128	Voor het Printen van de bal.
49425 C111	A9 3F	LDA-IMM	3F	# 63, voor 63 lussen.
49427 C113	05 FB	STA-Z.PAGE		Zet de teller.
49429 C115	20 B3EE	JSR-ABS	EEB3	Subr. DELAY, vertraag 1 ms.
49432 C118	06 FB	DEC-Z.PAGE		Verlaag de teller.
49434 C11A	D0 F9	BNE-REL	C115	Vertraag opnieuw.
49436 C11C	A9 20	LDA-IMM	20	# 32, voor CHR\$(32) (spatie).
49438 C11E	20 28C1	JSR-ABS	C128	Wis de bal (Print een spatie).
49441 C121	08	INY-IMPL		Cursor 1 Plaats naar rechts.
49442 C122	00 38	CPY-IMM	28	Eind van de regel?
49444 C124	D0 E6	BNE-REL	C10C	Voor het Printen op de volgende Plaats.
49446 C126	F0 E2	BEQ-REL	C10A	Begin opnieuw aan dezelfde regel.
49448 C128	48	PHA-IMPL		SUBROUTINE, save de inhoud van de accu.
49449 C129	18	CLC-IMPL		Carry bit 0.
49450 C12A	20 F0FF	JSR-ABS	FFF0	PLOT, cursor naar de juiste Plaats.
49453 C12D	68	PLA-IMPL		Haal de inhoud van de accu terug.
49454 C12E	20 D2FF	JSR-ABS	FFD2	CHROUT, Print het karakter.
49457 C131	60	RTS-IMPL		RETURN.

Wat tot nu toe nog niet is gebeurd dat is het veranderen van de kleur van de tekens. Dit gebeurt nu in dit programma in de twee eerste programmaregels, waarin de instructies hetzelfde effect hebben als PRINT CHR\$(5) voor witte tekens. Daarna wordt het scherm schoongemaakt met CLEAR SCREEN. Nu worden de gegevens ingevoerd voor het printen van de eerste bal. Eerst het regelnum-

mer \$ 02 in het X-register en de eerste plaats op die regel door kolomnummer \$ 00 in het Y-register.

Dan wordt de accu geladen met \$ 71 (113 decimaal) voor CHR\$(113).

Het printen zelf wordt uitgevoerd in de subroutine van dit programma.

Hierin wordt eerst de cursor op de juiste plaats gebracht met PLOT en daarna wordt het karakter geprint met CHROUT. Nu gebeurt alles in machinetaal zeer snel en de bal zal voordat hij wordt gewist toch nog enige tijd zichtbaar moeten zijn. Vandaar de vertraging die nu volgt. We gebruiken daarvoor de subroutine DELAY die het programma 1 ms ophoudt. In een lus die 63 keer wordt doorlopen wordt de subroutine ook 63 keer aangeroepen (regels 49425 tot en met 49434) en verkrijgen we voldoende vertraging om de bal met een redelijke snelheid over het scherm te zien gaan.

Direct na de vertraging wordt de bal gewist met PRINT CHR\$(32). Het X- en het Y-register hebben nog steeds dezelfde waarde zodat deze spatie komt op de plaats waar de bal geprint was. Hierna wordt de inhoud van het Y-register met 1 opgehoogd voor de volgende kolom. Is het einde van de regel nog niet bereikt dan wordt door de BNE in regel 49444 teruggegaan voor het printen van de volgende bal, maar nu op de volgende plaats van de regel. Dit gaat zo door totdat de bal op het eind van de regel is beland. Het Y-register heeft dan de waarde \$ 28 zodat door BEQ op regel 49446 naar regel 49418 wordt gesprongen en de bal opnieuw verschijnt aan het begin van de regel. De tijd tussen het wissen en het opnieuw printen van de bal is zo kort dat het verschil niet waarneembaar is. De snelheid van de bal over het scherm kunt u veranderen door voor \$ 3F in regel 49425 een ander getal te kiezen.

Het is ook mogelijk om de bal schuin over het scherm te laten gaan. In dat geval moet niet alleen de waarde van het Y-register worden veranderd, maar ook de waarde van het X-register.

Nu kan het ook gebeuren dat een bal tegen een voorwerp kaatst. In het volgende programma raakt een bal een muur en gaat daarna langs de zelfde weg weer terug. Hierbij doen zich twee problemen voor: hoe laten we de bal op een zo eenvoudig mo-

gelijke manier zowel heen als terug gaan en hoe stellen we vast of de bal de muur raakt.

Om de bal langs een regel te laten verplaatsen van links naar rechts moet de inhoud van het Y-register steeds met 1 worden verhoogd.

Het zal duidelijk zijn dat voor de tegenovergestelde richting de inhoud van het Y-register steeds met 1 zal moeten worden verlaagd. Dit is in dit programma verwezenlijkt door er steeds -1 bij op te tellen.

Hiervoor wordt een hulpregister (\$ OOFB) gebruikt dat met \$ 01 wordt geladen als de bal naar rechts moet gaan en met \$ FF wordt geladen (komt overeen met -1) als de bal in tegenovergestelde richting moet.

De inhoud van dit register wordt bij elke verplaatsing van de bal opgeteld bij het Y-register.

Om vast te stellen of de bal de muur heeft geraakt gaan we na of op de plaats, volgend op die van de bal, zich in het schermgeheugen de schermcode voor het muurkarakter bevindt. Nu zal de cursor na het printen van de bal zelf al één plaats naar rechts zijn gegaan, zodat zijn plaats niet meer overeenkomt met de plaats die door de waarden in het X- en het Y-register wordt aangegeven. De plaats waar de cursor zich bevindt correspondeert met een adres in het schermgeheugen en er is erg eenvoudig achter dit adres te komen.

Het adres van de betreffende geheugenplaats wordt aangegeven door de inhoud van de adressen \$ 00D1, \$ 00D2 en \$ 00D3. Op de adressen \$ 00D1 en \$ 00D2 bevinden zich de lage en de hoge byte van het schermregeladres waar zich de cursor bevindt. Het schermregeladres is het adres van de eerste geheugenplaats van de betreffende regel. In \$ 00D3 is het kolomnummer opgeslagen. Nu is de plaats van de cursor in het schermgeheugen eenvoudig te berekenen: het schermregeladres plus het kolomnummer! In het volgende programma is dat verwezenlijkt:

Kaatsende bal

49408 C100	20	44E5	JSR-ABS	E544	CLEAR SCREEN. voor CLR HOME.
49411 C103	A9	96	LDA-IMM	96	# 150, voor CHR\$(150).
49413 C105	20	D2FF	JSR-ABS	FFD2	Voor licht rode tekens.
49416 C108	A9	A1	LDA-IMM	A1	# 161, voor CHR\$(161).
49418 C10A	A0	25	LDY-IMM	25	Kolomnummer 37.
49420 C10C	A2	08	LDX-IMM	08	Regelnummer 8.
49422 C10E	20	51C1	JSR-ABS	C151	Print de muur.
49425 C111	E8		INX-IMPL		Verhoog het regelnummer.
49426 C112	E0	0F	CPX-IMM	0F	Einde van het Printen?
49428 C114	D0	F8	BNE-REL	C10E	Voor het volgende muurkarakter.

49430	C116	A9 05	LDA-IMM	05	Voor CHR\$(5).
49432	C118	20 D2FF	JSR-ABS	FFD2	Voor witte tekens.
49435	C11B	A9 01	LDA-IMM	01	
49437	C11D	85 FB	STA-Z.PAGE		Voor het verhogen van het kolomnummer.
49439	C11F	A2 0B	LDX-IMM	0B	De bal op regel nummer 12.
49441	C121	A0 00	LDY-IMM	00	De bal aan het begin van de regel.
49443	C123	A9 71	LDA-IMM	71	CHR\$(113), het karakter voor de bal.
49445	C125	20 51C1	JSR-ABS	C151	Print de bal.
49448	C128	A9 3F	LDA-IMM	3F	Voor 63 lussen.
49450	C12A	85 FC	STA-Z.PAGE		Zet de teller.
49452	C12C	20 B3EE	JSR-ABS	EEB3	DELAY, vertraag 1 ms.
49455	C12F	C6 FC	DEC-Z.PAGE		Verlaag de teller.
49457	C131	D0 F9	BNE-REL	C12C	Voor de volgende vertraging van 1 ms.
49459	C133	A9 20	LDA-IMM	20	Voor CHR\$(32), spatie.
49461	C135	20 51C1	JSR-ABS	C151	Wis de bal.
49464	C138	98	TYA-IMPL		Kolomnummer in ACCU.
49465	C139	48	PHA-IMPL		Save het kolomnummer.
49466	C13A	A4 D3	LDY-Z.PAGE		Plaats van de cursor op de regel.
49468	C13C	B1 D1	LDA-(IND),Y		Karakter onder de cursor in ACCU.
49470	C13E	C9 61	CMP-IMM	61	# 97, schermcode muurkarakter?
49472	C140	F0 09	BEQ-REL	C14B	SProng als de muur is geraakt.
49474	C142	68	PLA-IMPL		Kolomnummer in de ACCU.
49475	C143	18	CLC-IMPL		
49476	C144	65 FB	ADC-Z.PAGE		Corrigeer het kolomnummer.
49478	C146	A8	TAY-IMPL		Kolomnummer in Y-register.
49479	C147	10 DA	BPL-REL	C123	Voor de volgende afbeelding.
49481	C149	30 D0	BMI-REL	C11B	Bal aan het begin van de regel, opnieuw.
49483	C14B	A9 FF	LDA-IMM	FF	# 255, gelijk aan -1.
49485	C14D	85 FB	STA-Z.PAGE		Voor het verlagen van het kolomnummer.
49487	C14F	D0 F1	BNE-REL	C142	Voor het teruggaan van de bal.
49489	C151	48	PHA-IMPL		SUBROUTINE, save de inhoud van de ACCU.
49490	C152	18	CLC-IMPL		
49491	C153	20 F0FF	JSR-ABS	FFF0	PLOT, cursor naar de juiste plaats.
49494	C156	68	PLA-IMPL		Haal de inhoud van de ACCU terug.
49495	C157	20 D2FF	JSR-ABS	FFD2	CHROUT, Print het karakter.
49498	C15A	60	RTS-IMPL		RETURN.

Nadat het scherm wordt gewist wordt eerst de muur getekend. Dit doen we met licht rode tekens. Het kolomnummer is 37 en het eerste regelnummer is 8. Door dit regelnummer steeds met 1 te verhogen worden de karakters in kolom 37 onder elkaar geprint. CHR\$(161) geeft een verticale lijn. Nadat CHR\$(5) voor witte tekens is ingevoerd wordt geheugenplaats \$ 00FB met \$ 01 geladen voor het *verhogen* van het kolomnummer. De bal wordt eerst geschreven aan het begin van regelnummer 12 en na een vertraging (regels 49448 tot en met 49457) weer met CHR\$(32) gewist. Na het saven van het kolomnummer (dit geeft niet meer de plaats aan waar de cursor zich bevindt) laden we het *juiste* kolomnummer uit register \$ 00D3 in het Y-register. Met LDA(\$ 00D1),Y wordt de schermcode van het karakter onder de cursor in de accu geladen. Dit is de indirecte adressering waarbij het adres wordt gevonden door de inhoud van \$ 00D1 en \$ 00D2 te vermeerderen met de waarde van het Y-register. Door te vergelijken met \$ 61

kunnen we vaststellen of het een muurkarakter betreft. Is dat niet het geval dan wordt het kolomnummer verhoogd door het hierbij optellen van de inhoud van geheugenplaats \$ 00FB, die met \$ 01 is geladen. Betrof het echter wel een muurkarakter dan wordt in de regels 49483 tot en met 49487 de inhoud van \$ 00FB gelijk aan \$ FF voor -1. Dat zal tot gevolg hebben dat het kolomnummer in de regels 49474 tot en met 49478 zal worden verlaagd. Als laatste van het hoofdprogramma wordt vastgesteld of de bal al of niet weer aan het begin van de regel is aangeland. In dat geval wordt opnieuw begonnen met het verhogen van het kolomnummer. De snelheid van de bal is te veranderen door het getal \$ 3F in regel 49448 te wijzigen.

9.3. Het gebruik van de sprites

Over het gebruik van de sprites valt weinig nieuws mede te delen.

Evenals in BASIC betekent het niets meer dan het vullen van de betreffende geheugenplaatsen met

data, terwijl voor het bewegen van de sprites de inhoud van de registers voor de plaats van de sprites op het scherm voortdurend moeten worden veranderd. Dit moet dan wel met de nodige vertraging

geschieden omdat de snelheid van de sprites over het beeldscherm anders veel te groot is. Het volgende programma laat een kikker verticaal over het beeldscherm bewegen.

Bewegende sprite

49408 C100	20	44E5	JSR-ABS	E544	CLEAR SCREEN, voor CLR HOME.
49411 C103	A9	01	LDA-IMM	01	Voor wit veld.
49413 C105	8D	21D0	STA-ABS	D021	Register voor de veldkleur.
49416 C108	A2	3E	LDX-IMM	3E	Voor 63 geheugenplaatsen.
49418 C10A	BD	4BC1	LDA-ABS,X	C14B	Data in de ACCU.
49421 C10D	9D	4003	STA-ABS,X	0340	Data in het sPrite register.
49424 C110	CA		DEX-IMPL		Verlaag datatellen.
49425 C111	10	F7	BPL-REL	C10A	Voor volgende data.
49427 C113	A9	01	LDA-IMM	01	Zet de sPrite aan.
49429 C115	8D	15D0	STA-ABS	D015	sPrite enable register.
49432 C118	A9	0D	LDA-IMM	0D	# 13, vector, 13*64=832 (\$ 0340).
49434 C11A	8D	F807	STA-ABS	07F8	Vector register sPrite 0.
49437 C11D	A9	05	LDA-IMM	05	Voor de groene kleur.
49439 C11F	8D	27D0	STA-ABS	D027	sPrite 0 kleurregister.
49442 C122	A9	96	LDA-IMM	96	# 150 voor de X-waarde.
49444 C124	8D	00D0	STA-ABS	D000	sPrite 0 register X-waarde.
49447 C127	A0	32	LDY-IMM	32	# 50, sPrite bovenaan het scherm.
49449 C129	9C	01D0	STY-ABS	D001	sPrite 0 register Y-waarde.
49452 C12C	20	42C1	JSR-ABS	C142	Vertraag.
49455 C12F	C8		INY-IMPL		Verhoog de Y-waarde, sPrite naar omlaag.
49456 C130	C0	E7	CPY-IMM	E7	sPrite onderaan het scherm?
49458 C132	D0	F5	BNE-REL	C129	Voor de volgende Y-Positie.
49460 C134	88		DEY-IMPL		Verlaag de Y-waarde, sPrite naar boven.
49461 C135	8C	01D0	STY-ABS	D001	sPrite 0 register Y-waarde.
49464 C138	20	42C1	JSR-ABS	C142	Vertraag.
49467 C13B	C0	32	CPY-IMM	32	sPrite bovenaan het scherm?
49469 C13D	D0	F5	BNE-REL	C134	Voor de volgende Y-Positie.
49471 C13F	4C	29C1	JMP-ABS	C129	Herhaal de beweging.
49474 C142	A2	1F	LDX-IMM	1F	SUBROUTINE, voor 30 ms vertraging.
49476 C144	30	B3EE	JSR-ABS	EEB3	DELAY, vertraag 1 ms.
49479 C147	CA		DEX-IMPL		X=X-1.
49480 C148	D0	FA	BNE-REL	C144	Voor de volgende ms vertraging.
49482 C14A	60		RTS-IMPL		RETURN.
49483 C14B	00	3C 00 E0 7E 07 23 FF C4 36	DATA		
49493 C155	FF	6C 1C FF 38 18 FF 18 00 FF	DATA		
49503 C15F	00	00 FF 00 00 7E 00 00 7E 00	DATA		
49513 C169	00	FF 00 07 FF E0 0F 18 FC 0C	DATA		
49523 C173	00	30 0C 00 30 0C 00 30 0C 00	DATA		
49533 C17D	30	0C 00 30 E0 00 0F 00 00 00	DATA		
49543 C187	00	00 00	DATA		

De eerste regel van het programma zal u wel bekend voorkomen. De volgende twee regels brengen iets nieuws. Hier wordt het register voor de veldkleur met \$ 01 geladen zodat het veld op uw scherm wit wordt.

Dit is nodig om de groene kikker goed te laten afsteken. Op de zelfde manier kunt u eventueel ook de kleur van het kader bepalen. Nu volgt het laden van de sprite data registers met de gegevens die de vorm van de sprite bepalen. Dit komt in principe overeen met het READ statement. De data vinden

we direct achter het programma op de regels 49483 tot en met 49540. Met de absoluut, X adressering worden de sprite data registers (te beginnen met \$ 0340 + \$ 3E = \$ 037E) ingeschreven.

Nadat de sprite is 'aangezet' (register \$ D015) wordt de vector 13 (\$ 0D) geplaatst in het vectorregister voor sprite 0. Dit geeft aan waar de data voor de sprite 0 te vinden is. Nu worden achtereenvolgens de kleur (groen, adres \$ D027), de X-waarde (150, register \$ D000) en de Y-waarde (32, register \$ D001) ingevoerd. De sprite bevindt

zich nu bovenaan het scherm maar gaat zakken door het veranderen van het Y-waarde.

Na een verandering wordt steeds voor 30 ms vertraagd. Door de duur van deze vertraging te veranderen kan de snelheid van de sprite over het scherm worden ingesteld. Is de sprite onderaan het scherm aangeland (als de Y-waarde \$ E7 is) dan wordt de waarde van Y steeds verminderd. Hierdoor gaat de sprite weer omhoog tot hij bovenaan het veld is. Dan herhaalt het geheel zich.

U ziet, afgezien van de vertraging van 30 ms is er niets nieuws onder de zon en kunt u de programma's voor de sprites geheel overeenkomstig de BASIC programma's hiervoor samenstellen.

9.4. Scrolling

Het scrollen van het schermbeeld kennen we al: als we de laatste regel van het scherm hebben vol getypt dan schuift het gehele beeld één regel omhoog. Dit type scrolling is nogal grof en het type scrolling dat we nu gaan demonstreren is heel wat geleidelijker. Het beeld schuift daarbij precies één stip (het achtste deel van een karakter breedte of hoogte) per keer op. Deze scrolling is in alle vier de richtingen (horizontaal en verticaal) mogelijk. We beperken ons tot de horizontale scrolling naar links.

Belangrijk bij horizontale scrolling is het register \$ DO16 en wel de vier laagste bits hiervan. Als het getal in de drie laagste bits van dit register toeneemt van 0 tot en met 7, zien we het beeld in het veld in stapjes van elk een achtste karakterbreedte verplaatsen naar rechts en bij het verminderen van 7 naar 0 (0 is de normale situatie) verschuift het beeld weer naar links. Het kader blijft op zijn plaats. Met bit 3 kan de breedte van het kader worden ingesteld op 40 kolommen (normaal, bit 3 is 1) of 38 kolommen (bit 3 is 0).

Het beeld verandert niet, dat wil zeggen dat in de 38 kolommode (bit 3 is 0) bij de normale stand van het beeld zowel links als rechts een kolom wordt afgedekt door het kader. Dit geeft ons de mogelijkheid kolom 0 of kolom 39 ongezien in te schrijven.

De werkwijze bij scrolling naar links is als volgt:

- a. De scherm inhoud wordt over één kolom naar *rechts* verplaatst. (de bits 0 tot en met 2 van geheugenplaats \$ D016 worden '1').
- b. De tekst op het scherm wordt één kolom naar *links* verplaatst door elk teken in het schermgeheugen, per schermregel, in een lagere geheugenplaats te schrijven. Deze verplaatsing dient

voor het oog onzichtbaar snel te gebeuren zodat de handelingen a en b samen het effect geven van een beeld dat op zijn plaats blijft.

- c. Nu wordt betrekkelijk langzaam de scherm inhoud één kolom naar links verplaatst in kleine stapjes, zodat de verplaatsing geleidelijk lijkt te gebeuren. Dit geschiedt door het getal in de bits 0 tot en met 2 van geheugenplaats \$ DO16 (7) steeds met 1 te verlagen tot 0.

Het hierna volgende programma laat een sterrenbeeld over uw scherm bewegen. Omdat het programma nog al uitgebreid is zal het in gedeelten worden behandeld. Het geeft dan tevens de mogelijkheid om te laten zien op welke wijze een programma kan worden ontwikkeld en uitgevoerd.

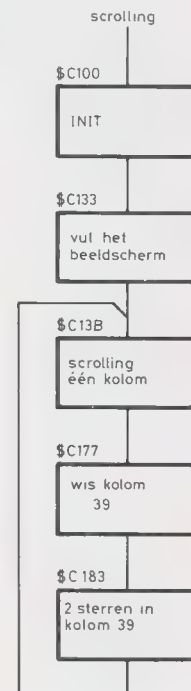


Fig. 44

Eerst wordt een stroomdiagram gemaakt van de hoofddelen (figuur 44). Vooraf moeten een aantal registers worden ingeschreven met een bepaalde waarde (kleur van het scherm en de tekens bijvoorbeeld). Verder moet een getal worden ingevoerd voor het genereren van een randomgetal voor het plaatsen van de sterren. Dit geschiedt aan het begin van het programma en wordt het *initie-ren* (INIT) genoemd. Dan moet het beeldscherm gevuld worden met sterren. Nu volgt de scrolling over één kolom van het scherm. Als laatste moet in

kolom 39 weer nieuwe sterren (twee stuks) worden geplaatst. Daartoe moet de kolom eerst worden gewist zodat alléén deze sterren in de kolom staan en niet ook anderen die daar reeds aanwezig waren. De sterren die nu in deze kolom zijn geplaatst worden zichtbaar bij de volgende scrolling.

De tweede stap bij het ontwikkelen van het programma is het uitwerken van de afzonderlijke blokken. Eerst het blok INIT. Het plaatsen van de sterren door middel van een print routine geeft moeilijkheden bij de laatste kolom, omdat na de printopdracht de cursor naar het begin van de volgende regel gaat. Vooral als een ster toevallig in de laatste geheugenplaats van de laatste kolom moet worden geplaatst. Dan vindt automatisch de scrolling van één regel omhoog plaats en dat is nu juist niet de bedoeling. Daarom moet het printen van de sterren gebeuren overeenkomstig het plaatsen van de schermcode in het schermgeheugen middels een POKE opdracht. Dit heeft consequenties voor het invoeren van de kleur van de karakters en het veld. Verder wordt in dit blok het getal 25 ingevoerd om later in het programma te worden gebruikt voor het genereren van $\text{INT}(25 \cdot \text{RND}(p))$. Dit kan een getal geven van 0 tot en met 24 voor het regelnummer.

Het tweede blok van figuur 44 is verder uitgewerkt in figuur 45. Dit programmadeel heeft het vullen van het beeldscherm met sterren tot doel. Voor het printen van een ster moet voor de plaats van die

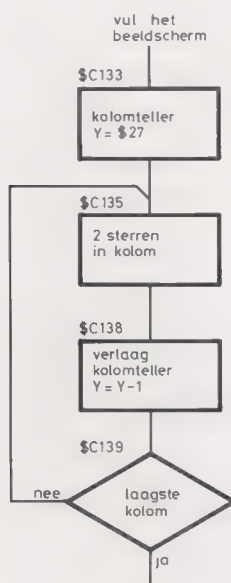


Fig. 45

ster het kolomnummer in het Y-register en het regelnummer in het X-register worden geladen. Het regelnummer wordt bij toeval bepaald in het blok 'plaats twee sterren in een kolom'. Door het verlagen van het kolomnummer in een lus wordt in elke kolom (behalve in kolom 0) willekeurig twee sterren geplaatst.

Het programmadeel 'scrolling over één kolom' is verder uitgewerkt in figuur 46. Dit deel is de kern van het programma en is geheel in overeenstem-

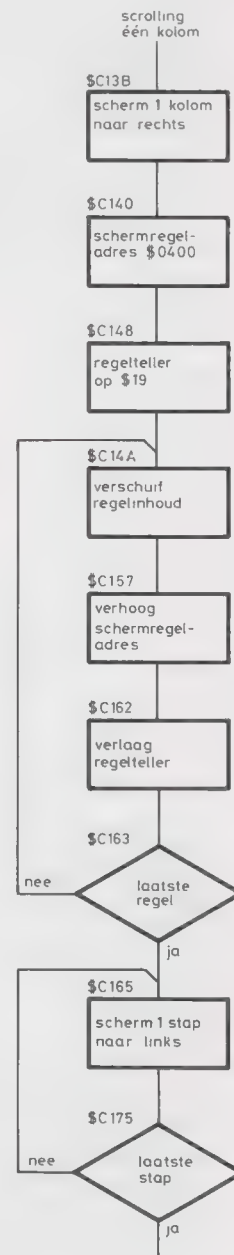


Fig. 46

ming met de hiervóór genoemde punten a, b en c. Het bovenste blok betreft punt a, de volgende zes blokken punt b en de laatste twee blokken punt c. Voor punt c wordt het regelnummer ingevoerd om (indirect), Y adressering mogelijk te maken. Met het kolomnummer in het Y-register wordt voor de eerste regel (schermregeladres \$ 0400) het adres van de van de geheugenplaats \$ 0400+Y. Voor elke volgende regel wordt het schermregeladres met \$ 28 (40) verhoogd.

De laatste blokken van figuur 44 zijn verder uitgewerkt in figuur 47. Eerst wordt het Y-register geladen met \$ 27 voor kolom 39 en het X-register met \$ 18 voor regelnummer 24. Nu wordt een spatie geprint. Het regelnummer (X) wordt verlaagd en er wordt een tweede spatie geprint. Dit gaat zo door totdat regel 0 aan de beurt is geweest. Daarna worden er twee sterren in de kolom geprint. Deze routine zijn we al eens eerder tegen gekomen (figuur 45) zodat deze als een subroutine zal worden uitgevoerd. In deze subroutine zal ook het printen van een karakter voorkomen zodat ook dit als subroutine zal worden uitgevoerd (ook nodig voor het printen van een spatie).

De derde stap bij het ontwikkelen van het programma zal bestaan uit het vaststellen van de juiste instructies. Om het geheel overzichtelijk te houden zal het programma dat zodoende wordt gevormd ook in gedeelten worden gegeven. Het eerste deel van het programma betreft uiteraard het blok INIT.

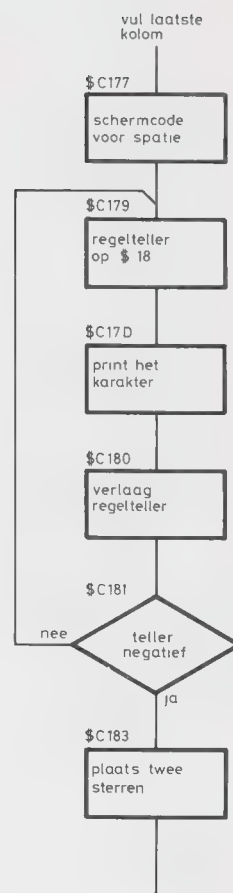


Fig. 47

Scrolling, INIT

49408 C100 A9 01 LDA-IMM	01 Voor witte kleur.
49410 C102 8D 21D0 STA-ABS	D021 Veldkleurregister.
49413 C105 20 44E5 JSR-ABS	E544 CLEAR SCREEN, voor CLR HOME.
49416 C108 A9 00 LDA-IMM	00 Voor zwart veld.
49418 C10A 8D 21D0 STA-ABS	D021 Veldkleurregister.
49421 C10D 8D 0EDC STA-ABS	DC0E Schakel interrupt uit.
49424 C110 A9 C0 LDA-IMM	C0 # 247, bit 3 is "0".
49426 C112 8D 16D0 STA-ABS	D016 Voor de 38 kolom mode.
49429 C115 A9 32 LDA-IMM	32 CHR\$(50) voor "2".
49431 C117 8D 00C8 STA-ABS	C800 "2" in de buffer.
49434 C11A A9 35 LDA-IMM	35 CHR\$(53), voor "5".
49436 C11C 8D 01C8 STA-ABS	C801 "5" in de buffer.
49439 C11F A9 02 LDA-IMM	02 Voor 2 karakters.
49441 C121 A2 00 LDX-IMM	00 La9e byte bufferadres.
49443 C123 86 22 STX-Z.PAGE	Pointer laa9.
49445 C125 A2 C8 LDX-IMM	C8 Hoge byte bufferadres.
49447 C127 86 23 STX-Z.PAGE	Pointer hoo9.
49449 C129 20 B5B7 JSR-ABS	B7B5 Van ASCII naar FLOATING POINT in FPAC.
49452 C12C A2 00 LDX-IMM	00 La9e-byte variabele X.
49454 C12E A0 C9 LDY-IMM	C9 Hoge byte variabele X.
49456 C130 20 D4BB JSR-ABS	BBD4 Getal in de geheugenruimte voor X.

De eerste regels betreffen het invoeren van de kleur. Eerst wordt het veld wit door \$ 01 in het veldkleurregister te schrijven. Direct daarna volgt CLEAR SCREEN. Dit heeft tot gevolg dat niet alleen het beeldscherm wordt gewist maar dat ook de hierna ingevoerde karakters wit worden geprint. Met het witte veld als achtergrond zouden ze niet te zien zijn. Nu wordt het veld echter weer zwart door het schrijven van \$ 00 in het veldkleurregister.

Dit soort scrolling programma's kunnen storing ondervinden door de interrupts voor het lezen van

het toetsenbord. Reden waarom in regel 49421 de interrupt wordt uitgeschakeld door \$ 00 in register \$ DCOE te schrijven.

Het register \$ DO16 is in normale situaties gevuld met \$ C8.

Voor de 38 kolommode wordt hierin \$ CO geschreven zodat bit 3 nul wordt.

De rest van dit programmeel wordt gebruikt voor het invoeren van het getal 25 voor het bepalen van INT(25*RND(p)).

Het volgende programmeel is afgeleid van figuur 45.

Vul het beeldscherm

49459 C133	A0 27	LDY-IMM	27	Voor 39 kolommen.
49461 C135	20 89C1	JSR-ABS	C189	Plaats twee sterren in een kolom.
49464 C138	88	DEY-IMPL		Verlaag kolommenteller.
49465 C139	D0 FA	BNE-REL	C135	Voor de volgende kolom.

De instructies volgen geheel uit het figuur en behoeven verder geen toelichting.

Het derde programmeel is de uitwerking van figuur 46.

Scrolling 1 kolom

49467 C13B	A9 C7	LDA-IMM	C7	Bit 3 blijft 0.
49469 C13D	8D 16D0	STA-ABS	D016	Beeld op scherm 1 kolom naar rechts.
49472 C140	A9 00	LDA-IMM	00	Laag byte schermregeladres.
49474 C142	85 FB	STA-Z.PAGE		Laag byte vector.
49476 C144	A9 04	LDA-IMM	04	Hoog byte schermregeladres.
49478 C146	85 FC	STA-Z.PAGE		Hoog byte vector.
49480 C148	A2 19	LDX-IMM	19	Regelteller voor 25 regels.
49482 C14A	A0 01	LDY-IMM	01	Voor het tweede karakter op de regel.
49484 C14C	B1 FB	LDA-(IND),Y		Haal het karakter op.
49486 C14E	88	DEY-IMPL		Ga 1 kolom terug.
49487 C14F	91 FB	STA-(IND),Y		Karakter in geheugenplaats.
49489 C151	C8	INY-IMPL		1 kolom verder.
49490 C152	C8	INY-IMPL		Naar 1 kolom verder.
49491 C153	C0 28	CPY-IMM	28	Voor 40 kolommen.
49493 C155	D0 F5	BNE-REL	C14C	Voor het volgende karakter.
49495 C157	A9 28	LDA-IMM	28	40 geheugenplaatsen per regel.
49497 C159	18	CLC-IMPL		Carry is "0".
49498 C15A	65 FB	ADC-Z.PAGE		Laag byte + 40.
49500 C15C	85 FB	STA-Z.PAGE		Laag byte vector.
49502 C15E	90 02	BCC-REL	C162	Spring indien geen overdracht.
49504 C160	E6 FC	INC-Z.PAGE		Hoog byte schermregeladres + 1.
49506 C162	CA	DEX-IMPL		Verlaag de regelteller.
49507 C163	D0 E5	BNE-REL	C14A	Voor een volgende regel.
49509 C165	CE 16D0	DEC-ABS	D016	Beeld op het scherm 1 stapje naar links.
49512 C168	A2 1F	LDX-IMM	1F	Voor 30 ms.
49514 C16A	20 B3EE	JSR-ABS	EEB3	DELAY, vertraag 1 ms.
49517 C16D	CA	DEX-IMPL		X=X-1
49518 C16E	D0 FA	BNE-REL	C16A	Voor de volgende ms vertraaging.
49520 C170	A9 C0	LDA-IMM	C0	Voor het laatste stapje.
49522 C172	CD 16D0	CMF-ABS	D016	Ga na of laatste stapje is voltooid.
49525 C175	D0 EE	BNE-REL	C165	Voor volgend stapje.

Alle kolommen op het scherm worden in één keer naar rechts verplaatst door het laden van geheugenplaats \$ DO16 met \$ C7. Daarna wordt het schermregeladres \$ 0400 in \$ FB en \$ FC geplaatst zodat in regel 49484 met LDA(IND), Y het karakter uit plaats 1 van de regel wordt gelezen (Y = \$ 01) en in plaats 0 kan worden geschreven (Y = \$ 00 door DEY).

Door twee keer INY wordt Y \$ 02 en kan de volgende geheugenplaats worden overgebracht. Dit gaat door tot Y = \$28 (40, regel 49491).

Daarop wordt het schermregeladres op gehooft met \$ 28 en wordt de volgende regel behandeld. Register X is de regelteller. Als deze nul is dan is

het gehele scherm één kolom naar links gebracht (regel 49507).

In regel 49509 gaat de inhoud van het scherm één stapje naar links door het verlagen van de inhoud van \$ DO16. Nu is een vertraging van 30 ms nodig vóórdat opnieuw het volgende stapje wordt gedaan. Dit is bepalend voor de snelheid van de scrolling over het scherm en deze kan geregeld worden met het getal na LDX in regel 49512. Omdat de laatste kolom van het scherm slechts is uitgelezen en niet daarna is ingeschreven vinden we hierin nog twee sterren. Voordat hierin twee nieuwe sterren worden geplaatst moet deze kolom daarom worden gewist (figuur 47).

Vul kolom 39

49527	C177	A9 20	LDA-IMM	20	# 32, schermcode voor spatie.
49529	C179	A2 18	LDX-IMM	18	Voor 25 regels.
49531	C17B	A0 27	LDY-IMM	27	Kolomnummer 39.
49533	C17D	20 ABC1	JSR-ABS	C1AB	Print (POKE) het karakter.
49536	C180	CA	DEX-IMPL		Verlaag de regelteller.
49537	C181	10 FA	BPL-REL	C17D	Voor de volgende regel.
49539	C183	20 89C1	JSR-ABS	C189	Plaats twee sterren in kolom 39.
49542	C186	4C 3BC1	JMP-ABS	C13B	Voor de volgende scrolling.

In dit programmadeel worden twee subroutines aangeroepen, eerst de subroutine voor het printen van het karakter \$ 20 (spatie) op regel 49533 voor het wissen van kolom 39 (Y = \$ 27) en later de subroutine voor het printen van twee sterren in deze kolom. De instructies zijn in overeenstem-

ming met figuur 47. Als laatste wordt teruggegaan naar het begin van het programma, achter het blok INIT, voor de volgende scrolling.

Het programma wordt compleet gemaakt met de volgende subroutines:

Scrolling, subroutines

49545	C189	A2 02	LDX-IMM	02	Teller, voor 2 sterren.
49547	C18B	8A	TXA-IMPL		
49548	C18C	48	PHA-IMPL		Save de teller.
49549	C18D	98	TYA-IMPL		
49550	C18E	48	PHA-IMPL		Save het kolomnummer.
49551	C18F	20 97E0	JSR-ABS	E097	RND, randomgetal in FPAC.
49554	C192	A9 00	LDA-IMM	00	Laag byte variabele X.
49556	C194	A0 C9	LDY-IMM	C9	Hoog byte variabele X.
49558	C196	20 28BA	JSR-ABS	BA28	MULTIPLY, X*RND(P) in FPAC.
49561	C199	20 9BBC	JSR-ABS	BC9B	INT, INT(X*RND(P)) in FPAC, regelnummer.
49564	C19C	68	PLA-IMPL		Haal het kolomnummer op.
49565	C19D	A8	TAY-IMPL		Kolomnummer in Y.
49566	C19E	A6 65	LDX-Z.PAGE		Regelnummer in X.
49568	C1A0	A9 2A	LDA-IMM	2A	# 42, schermcode voor asterisk.
49570	C1A2	20 ABC1	JSR-ABS	C1AB	Print (POKE) het karakter.
49573	C1A5	68	PLA-IMPL		Haal de teller op.
49574	C1A6	AA	TAX-IMPL		Teller in X.
49575	C1A7	CA	DEX-IMPL		Verlaag de teller.
49576	C1A8	D0 E1	BNE-REL	C18B	Voor de volgende ster.
49578	C1AA	60	RTS-IMPL		RETURN.
49579	C1AB	48	PHA-IMPL		Save het karakter.
49580	C1AC	18	CLC-IMPL		Carry "0".

49581 C1AD	20 F0FF	JSR-ABS	FFF0	Plaats de cursor.
49584 C1B0	68	PLA-IMPL		Haal het karakter weer op.
49585 C1B1	91 D1	STA-(IND),Y		Karakter in het schermgeheugen.
49587 C1B3	60	RTS-IMPL		RETURN.

Het eerste gedeelte is voor het printen van twee sterren in de kolom die door het Y-register wordt aangegeven. Hiervoor is een teller met de inhoud \$ 02 nodig. Deze teller en het kolomnummer moeten eerst in de stack worden bewaard omdat het X- en het Y-register nodig zijn voor het vormen van $INT(25 \cdot RND(p))$ in de regels 49551 tot en met 49561.

In dit programma komen slechts positieve getallen voor zodat de inhoud van FPAC (p) er niet toe doet. Voor de INT functie *moet* de subroutine op \$ BC9B worden gebruikt (niet die op \$BCCC) omdat later in het programma het resultaat uit \$ 0065 van FPAC in het X-register moet worden geladen voor het regelnummer (regel 49566). Intussen is ook het kolomnummer weer opgehaald en wordt het karakter (de ster) in regel 49568 in de betreffende geheugenplaats geladen. Het laatste gedeelte van de subroutine zorgt er voor dat deze twee maal wordt doorlopen.

Het printen van het karakter komt tot stand door een POKE functie. Het adres waar het karakter moet worden geplaatst wordt gevonden door eerst de cursor op de plaats te brengen die door het regelnummer in het X-register en het kolomnummer in het Y-register wordt aangegeven (regel 49581). Het adres van de overeenkomstige geheugenplaats in het schermgeheugen vinden we in de registers \$ 00D1 en \$ 00D2, in combinatie met het kolomnummer in het Y-register (hetzelfde als de inhoud van \$ 00D3). Daarom kan in regel 49585 het karakter in de juiste geheugenplaats worden gebracht.

9.5. Grafische functies met hoogoplossend vermogen

In principe is het mogelijk om voor elke toepassing die uw computer heeft bij BASIC programma's ook een machinetaalprogramma te schrijven. Dat geldt ook voor de bit map mode. Hierbij kan elke stip op het scherm apart worden aangestuurd. Het gebruik van de bit map mode bij BASIC programma's is uitgebreid behandeld in het boek 'leren programmeren met de commodore 64', door de Muiderkring uitgegeven.

Voor hen die niet met deze techniek op de hoogte zijn zullen de noodzakelijke punten ook in dit boek worden behandeld.

Het beeldscherm wordt gevormd uit een raster van 320 bij 200 stippen.

In de bit map mode wordt elke stip bestuurd door een bit in een geheugenplaats. Is dat bit 1 dan neemt de stip de voorgrondkleur aan, anders de achtergrondkleur. Dat betekent dat er $320 \cdot 200 = 64000$ bits nodig zijn. Dit zijn $64000/8 = 8000$ bytes. Dit veld van 8000 bytes wordt het bit map geheugen genoemd. De VIC II chip moet weten waar dit bit map geheugen aanvangt. Dit gaat op overeenkomstige wijze als het aanwijzen van de plaats van de karaktergenerator, met de bits 1 tot en met 3 van geheugenplaats \$ DO18 (53272). In deze paragraaf wordt als startadres van het bit map geheugen \$ 2000 gebruikt (8192) zodat het getal 4 in de genoemde bits van de geheugenplaats moet worden geschreven, zie hiervoor ook lijst 5. De inhoud van geheugenplaats \$ DO18 moet daarvoor \$ 19 (25) zijn.

De bit map mode moet worden ingeschakeld. Hiervoor moet bit 5 van geheugenplaats \$ DO11 '1' worden gemaakt. Dit is mogelijk met een OR bewerking tussen \$ 20 en de inhoud van de geheugenplaats.

Omdat in het bit map-geheugen acht bits zijn samengevoegd tot één byte zijn ook acht punten op het scherm samengevoegd tot een groepje.

Bij de karaktermode is de VIC II-chip gewend te werken met roosters van 8×8 punten en dat wordt ook hier aangehouden. De eerste acht bytes van het bit map-geheugen bedienen de 64 punten van het 8×8 rooster dat geheel links aan de bovenkant van het scherm is geplaatst. Er zijn daarom regels te herkennen (net als bij de karakters) met op elke regel 40 van die roosters. Voor elk rooster vinden we een blok van acht registers in het bit map-geheugen. Eerste rooster op regel 0: byte 0 tot en met byte 7 in de geheugenplaatsen 8192 tot en met 8199. Tweede rooster op regel 0: byte 8 tot en met byte 15 in de geheugenplaatsen 8200 tot en met 8207. Dit gaat zo regel voor regel door tot het laatste rooster in de rechter onderhoek (fig. 48).

Deze organisatie maakt het mogelijk het oorspronkelijk schermgeheugen als color-RAM te gebruiken.

ken voor elk overeenkomstig rooster. In elke byte van het schermgeheugen kan een getal worden geschreven waarvan de hoge tetraide de voorgrondkleur bepaalt (de enen in het bit map-geheugen) en de lage tetraide de achtergrondkleur, van het bijbehorende 8×8 rooster. Het getal dat in een register van het schermgeheugen moet worden geschreven kan worden gevonden door het codegetal van de voorgrondkleur te vermenigvuldigen met 16 en hier het codegetal van de achtergrondkleur bij op te tellen. De indeling van de roosters is hierbij gelijk aan die van de karaktermode.

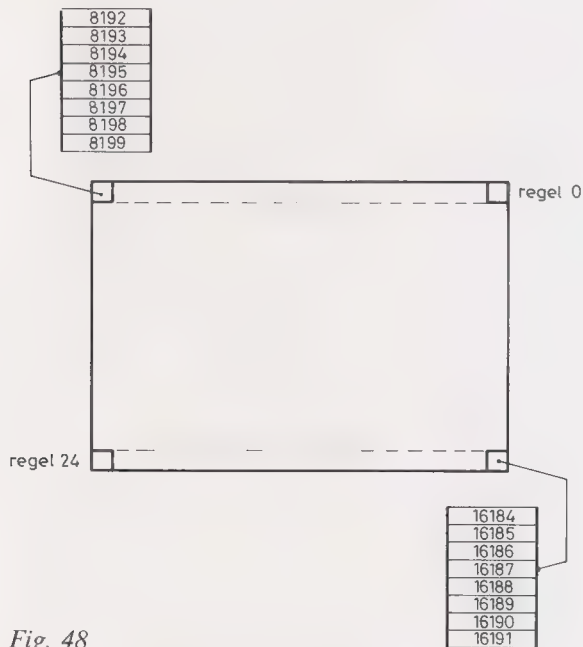


Fig. 48

Nu is het de bedoeling om ergens op het scherm een punt 'aan te zetten' dat wil zeggen dat ergens in het bit map-geheugen één bit 1 moet worden gemaakt. Het is nu de kunst te bepalen welk bit. Voor het teken van een figuur is het nodig dit bij herhaling te doen voor punten die naast elkaar liggen. Er worden dan lijnen gevormd. Elke punt kan worden georiënteerd met een X- en een Y-coördinaat. Om te beginnen kiezen we een punt in de linker bovenhoek (0.0). De X-waarde neemt toe naar rechts, de Y-waarde naar beneden. Dit is in principe in overeenstemming met de volgorde van de bytes in het geheugen.

In fig. 49 is een punt van een rooster zwart gemaakt. Deze punt wordt met 1 gekenmerkt. Hij bevindt zich op de plaats (X,Y) van het scherm.

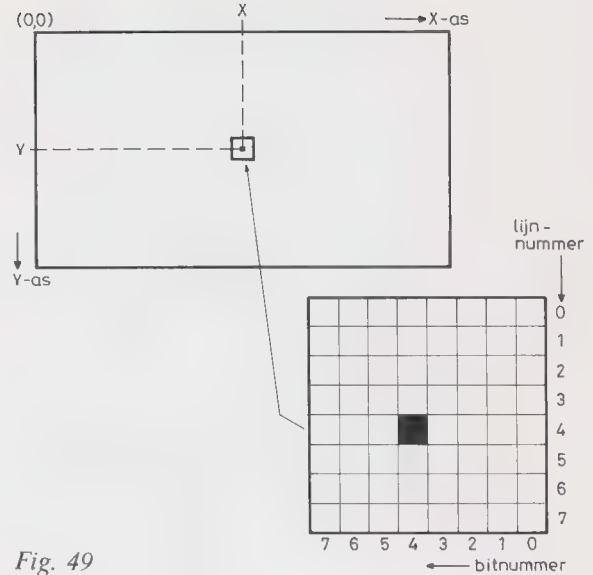


Fig. 49

We gaan eerst de plaats van het rooster bepalen waarin de punt voorkomt. De regel (R), waarop zich het rooster bevindt, is te vinden met:

$$R = \text{INT}(Y/8)$$

De kolom (K) waarin het rooster staat berekenen we met:

$$K = \text{INT}(X/8)$$

Een rooster bestaat uit 8 (horizontale) lijnen. De lijn in het rooster wordt weergegeven door de drie laatste bits van de Y-waarde. Het lijnnummer (L) is daarom te vinden met:

$$L = Y \text{ AND } 7$$

Hiermee wordt alleen de waarde van de drie laatste bits bepaald.

Er zijn acht bits op een lijn. Het bitnummer loopt echter naar links op terwijl de X-waarde naar rechts oploopt. Het bitnummer (BI) volgt daarom uit:

$$BI = 7 - (X \text{ AND } 7)$$

Nu kunnen we het nummer van de byte bepalen (BY) waarin zich de betreffende punt bevindt.

Ten eerste gaan we R regels naar beneden. Er zijn op elke regel $40 \times 8 = 320$ bytes (40 roosters van elk 8 bytes): $R \times 320$.

We gaan nu nog K kolommen (roosters) naar rechts op de regel. Elk rooster heeft 8 bytes: $K*8$. Nu zijn we aangekomen bij het rooster met de betreffende stip op lijn L.

$$BY = R*320 + K*8 + L$$

Het adres in bit map-geheugen is nu met het startadres hiervan (8192) vermeerderd met het aantal bytes:

$$AD = 8192 + R*320 + K*8 + L$$

Om nu het juiste bit op dit adres aan te zetten en alle anderen niet te veranderen (kan nodig zijn bij kruisende lijnen bijvoorbeeld) gebruiken we de volgende instructie.

POKE AD,PEEK(AD) OR 2 | BI

Dat de X-as niet als een denkbeeldig lijn midden over het scherm loopt maar door de computer geheel boven aan het scherm is gesitueerd, bevalt in de meeste gevallen niet. In fig. 50 is de X-as in het midden van het scherm gekozen (op lijn 100).

Vanaf het punt (0,0) is Y positief (tot maximum 100) naar boven en negatief naar onder. Voor het berekenen van het regelnummer en het lijnnummer geldt nu:

$$R = \text{INT}((100 - Y)/8)$$

$$L = (100 - Y) \text{ AND } 7$$

In verband met zijn omvang zal het volgende programma weer in delen worden gegeven. Het is een algemeen programma dat bij het tekenen van lijnen (grafieken, functies) kan worden gebruikt.

Hoge resolutie, INIT

49408 C100	A9 19	LDA-IMM	19	# 25.
49410 C102	8D 18D0	STA-ABS	D018	Startadres bit map geheugen \$ 2000.
49413 C105	A9 20	LDA-IMM	20	# 32.
49415 C107	0D 11D0	ORA-ABS	D011	Bit 5 wordt 1.
49418 C10A	8D 11D0	STA-ABS	D011	Bit map mode ingeschakeld.
49421 C10D	A0 00	LDY-IMM	00	Laagste byte startadres bit map geheugen.
49423 C10F	84 FB	STY-Z.PAGE		Pointer laag.
49425 C111	A2 20	LDX-IMM	20	Hoogste byte startadres bit map geheugen.
49427 C113	86 FC	STX-Z.PAGE		Pointer hoog.
49429 C115	A2 40	LDX-IMM	40	Laagste byte eindadres bit map geheugen.
49431 C117	86 FD	STX-Z.PAGE		LEAD.
49433 C119	A2 3F	LDX-IMM	3F	Hoogste byte eindadres bit map geheugen.
49435 C11B	86 FE	STX-Z.PAGE		HEAD.
49437 C11D	98	TYA-IMPL		\$ 00 in de ACCU.
49438 C11E	20 29C3	JSR-ABS	C329	Wis het bit map geheugen.
49441 C121	A2 04	LDX-IMM	04	Hoogste byte startadres kleur geheugen.
49443 C123	86 FC	STX-Z.PAGE		Pointer hoog.
49445 C125	A2 E8	LDX-IMM	E8	Laagste byte eindadres kleur geheugen.
49447 C127	86 FD	STX-Z.PAGE		LEAD.
49449 C129	A2 07	LDX-IMM	07	Hoogste byte eindadres kleur geheugen.
49451 C12B	86 FE	STX-Z.PAGE		HEAD.
49453 C12D	A0 00	LDY-IMM	00	Voor de eerste geheugenplaats.
49455 C12F	A9 10	LDA-IMM	10	Voor witte tekens op zwarte achtergrond.
49457 C131	20 29C3	JSR-ABS	C329	Vul het kleur geheugen.
49961 C329	A6 FC	LDX-Z.PAGE		Hoogste byte startadres.
49963 C32B	91 FB	STA-(IND),Y		Laadt de geheugenplaats.
49965 C32D	E4 FE	CPX-Z.PAGE		Paginanummer EAD (HEAD) bereikt?
49967 C32F	F0 08	BEO-REL	C339	Ja, vergelijk laagste byte EAD.
49969 C331	08	INY-IMPL		Nee, verhoog laagste byte.
49970 C332	D0 F7	BNE-REL	C32B	Voor volgende geheugenplaats.
49972 C334	E8	INX-IMPL		Verhoog hoogste byte.
49973 C335	86 FC	STX-Z.PAGE		Pointer hoog.
49975 C337	D0 F2	BNE-REL	C32B	Voor de volgende pagina.
49977 C339	08	INY-IMPL		Verhoog laagste byte.
49978 C33A	04 FD	CPY-Z.PAGE		Laatste geheugenplaats?
49980 C33C	D0 ED	BNE-REL	C32B	Voor de volgende geheugenplaats.
49982 C33E	60	RTS-IMPL		RETURN.

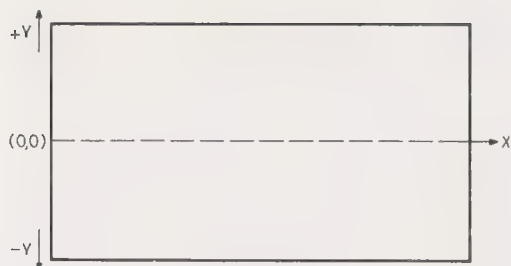


Fig. 50

Het eerste programmadeel is een gedeelte van het initiëren. Om te beginnen wordt het startadres van het bit map-geheugen ingevoerd door 25 (\$ 19) in te voeren in geheugenplaats \$ D018 (53272). Hiermee komt het getal 4 in de bits 1 tot en met 3, hetgeen adres \$ 2000 (8192) als startadres vastlegt. Vervolgens wordt de bit map mode ingeschakeld. Hiervoor wordt bit 5 van geheugenplaats \$ D011 (53265) '1' gemaakt met de OR functie OR \$ 20 (32). Nu volgt het wissen van het bit map-geheugen. Dit wissen gebeurt door de geheugenplaatsen met \$ 00 in te schrijven. De omvang van het bit map-geheugen is echter te groot (8000 geheugenplaatsen) om het met een index register geheel te kunnen adresseren. Daarom wordt van een indirect adres gebruik gemaakt. Bij aanvang is dat adres \$ 2000. Dit adres wordt als vector in de

geheugenplaatsen \$ 00FB en \$ 00FC geladen. Het eindadres van het bit map-geheugen (laatste adres + 1) wordt in de geheugenplaatsen \$ 00FD en \$ 00FE geladen. De adressen \$ 00FB tot en met \$ 00FE zijn overigens de enige adressen op pagina \$ 00 die geen bestemming hebben voor de BASIC interpreter of het systeemprogramma en ze zijn daardoor beschikbaar voor de gebruiker. Het eigenlijke werk doet de subroutine op adres \$ C329 (regel 49961). Deze laadt met de index,Y methode ($Y = \$ 00$, regel 49421) \$ 00 ($A = \$ 00$, regel 49437) in de geheugenplaatsen \$ 2000 tot en met \$ 20FF, de eerste pagina van het bit map geheugen (\$ 20). Daarna wordt het paginanummer verhoogd (regels 49972 en 49973). Dit paginanummer wordt in de eerste regel van de subroutine in het X-register geladen om met vergelijken te kunnen vaststellen of het laatste paginanummer al is bereikt (regel 49965). Is dit bereikt dan wordt ook van de lage byte van het adres (in het Y-register) voortdurend bepaald of het de hoogste waarde (\$ 40) heeft verkregen (regel 49978).

Op gelijke wijze wordt ook het kleurgeheugen voor de bit map mode, normaal het schermgeheugen, van zijn inhoud voorzien. Nu echter niet \$ 00 maar \$ 10 (16) voor de voorgrondkleur wit (hoge tetra-de van \$ 10) en de achtergrondkleur zwart (lage tetra-de).

Hoge resolutie, invoeren van constanten

49460	C134	A9 31	LDA-IMM	31	CHR\$(49) voor "1".
49462	C136	8D 00C8	STA-ABS	C800	"1" in de buffer.
49465	C139	A9 30	LDA-IMM	30	CHR\$(48) voor "0".
49467	C13B	8D 01C8	STA-ABS	C801	"0" in de buffer.
49470	C13E	8D 02C8	STA-ABS	C802	"0" in de buffer.
49473	C141	A9 03	LDA-IMM	03	Stringlengte, voor 3 karakters.
49475	C143	A2 0F	LDX-IMM	0F	Variabele adres (lage byte).
49477	C145	86 FD	STX-Z.PAGE		Variabele Pointer.
49479	C147	20 00C3	JSR-ABS	C300	Subroutine, getal in \$ CA0F.
49482	C14A	A9 37	LDA-IMM	37	CHR\$(55) voor "7".
49484	C14C	8D 00C8	STA-ABS	C800	"7" in de buffer.
49487	C14F	A9 01	LDA-IMM	01	Voor 1 karakter.
49489	C151	A2 19	LDX-IMM	19	Variabele adres.
49491	C153	86 FD	STX-Z.PAGE		Variabele Pointer.
49493	C155	20 00C3	JSR-ABS	C300	Getal in \$ CA19.
49496	C158	EE 00C8	INC-ABS	C800	"8" in de buffer.
49499	C15B	A9 01	LDA-IMM	01	Voor 1 karakter.
49501	C15D	A2 14	LDX-IMM	14	Variabele adres.
49503	C15F	86 FD	STX-Z.PAGE		Variabele Pointer.
49505	C161	20 00C3	JSR-ABS	C300	Getal in \$ c914.
49508	C164	A2 31	LDX-IMM	31	CHR\$(49) voor "1".
49510	C166	8E 01C8	STX-ABS	C801	"1" in de buffer.
49513	C169	A0 39	LDY-IMM	39	CHR\$(57) voor "9".
49515	C16B	8C 02C8	STY-ABS	C802	"9" in de buffer.
49518	C16E	E8	INX-IMPL		CHR\$(50) voor "2".
49519	C16F	8E 03C8	STX-ABS	C803	"2" in de buffer.

49522	C172	A9 04	LDA-IMM	04	Voor 4 karakters.
49524	C174	A2 1E	LDX-IMM	1E	Variabele adres.
49526	C176	86 FD	STX-Z.PAGE		Variabele Pointer.
49528	C178	20 00C3	JSR-ABS	C300	Getal "8192" in \$ C91E.
49531	C17B	A2 33	LDX-IMM	33	CHR\$(51), voor "3".
49533	C17D	8E 00C8	STX-ABS	C800	"3" in de buffer.
49536	C180	CA	DEX-IMPL		CHR\$(50), voor "2".
49537	C181	8E 01C8	STX-ABS	C801	"2" in de buffer.
49540	C184	A2 30	LDX-IMM	30	CHR\$(48), voor "0".
49542	C186	8E 02C8	STX-ABS	C802	"0" in de buffer.
49545	C189	A9 03	LDA-IMM	03	Voor 3 karakters.
49547	C18B	A2 23	LDX-IMM	23	Variabele adres.
49549	C18D	86 FD	STX-Z.PAGE		Variabele Pointer.
49551	C18F	20 00C3	JSR-ABS	C300	Getal in \$ C923.
49920	C300	A2 00	LDX-IMM	00	Lage byte bufferadres.
49922	C302	86 22	STX-Z.PAGE		Pointer laa9.
49924	C304	A2 C8	LDX-IMM	C8	Hoge byte bufferadres.
49926	C306	86 23	STX-Z.PAGE		Pointer hoo9.
49928	C308	20 B5B7	JSR-ABS	B7B5	Van ASCII naar FLOATING POINT in FPAC.
49931	C30B	A6 FD	LDX-Z.PAGE		Lage byte variabele adres.
49933	C30D	A0 C9	LDY-IMM	C9	Hoge byte variabele adres.
49935	C30F	20 D4BB	JSR-ABS	BBD4	FPAC in variabele adres.
49938	C312	60	RTS-IMPL		RETURN.

Het tweede gedeelte van het programma betreft ook het initiëren en dient voor het invoeren van de getallen (constanten) die bij het berekenen van het adres AD en het in te voeren byte nodig zijn.

Dit invoeren geschiedt op de bekende wijze (READ): eerst de karakters in de buffer, dan omzetting van floating point in FPAC en daarna het wegschrijven in het variabele-adres. De pointer

voor dat adres (de lage byte daarvan) wordt weggeschreven in \$ OOFD, zodat de subroutine op regel 49920 voor elk getal het voornaamste werk kan doen. Per getal zijn de lage byte van het adres (in \$ OOFD) en de lengte van de string in de buffer (in de accu) verschillend.

Het derde programmadeel berekent het adres AD en BI:

Berekening van het adres en BI

49983	C33F	A9 0F	LDA-IMM	0F	Lage byte getal 100.
49985	C341	A0 C9	LDY-IMM	C9	Hoge byte getal 100.
49987	C343	20 8CBA	JSR-ABS	BA8C	"100" in HLPREG.
49990	C346	A9 00	LDA-IMM	00	Lage byte variabele Y.
49992	C348	A0 C9	LDY-IMM	C9	Hoge byte variabele Y.
49994	C34A	20 A2BB	JSR-ABS	BBA2	Y in FPAC.
49997	C34D	20 53B8	JSR-ABS	B853	SUBTRACT, 100-Y in FPAC.
50000	C350	A2 AD	LDX-IMM	AD	Lage byte variabele adres.
50002	C352	A0 C9	LDY-IMM	C9	Hoge byte variabele adres.
50004	C354	20 D4BB	JSR-ABS	BBD4	100-Y in variabele adres.
50007	C357	20 0CB0	JSR-ABS	BC0C	FPAC naar HLPREG.
50010	C35A	A9 19	LDA-IMM	19	Lage byte getal 7.
50012	C35C	A0 C9	LDY-IMM	C9	Hoge byte getal 7.
50014	C35E	20 A2BB	JSR-ABS	BBA2	"7" in FPAC.
50017	C361	20 E9AF	JSR-ABS	AFE9	AND, (100-Y)AND7 in FPAC (L).
50020	C364	A2 28	LDX-IMM	28	Lage byte variabele adres voor A.
50022	C366	A0 C9	LDY-IMM	C9	Hoge byte variabele adres voor A.
50024	C368	20 D4BB	JSR-ABS	BBD4	L in A.
50027	C36B	A9 14	LDA-IMM	14	Lage byte getal 8.
50029	C36D	A0 C9	LDY-IMM	C9	Hoge byte getal 8.
50031	C36F	20 A2BB	JSR-ABS	BBA2	"8" in FPAC.
50034	C372	A9 AD	LDA-IMM	AD	Lage byte variabele 100-Y.
50036	C374	A0 C9	LDY-IMM	C9	Hoge byte variabele 100-Y.
50038	C376	20 8CBA	JSR-ABS	BA8C	100-Y in HLPREG.
50041	C379	20 0CB0	JSR-ABS	BC0C	DIVIDE, (100-Y)/8 in FPAC.
50044	C37C	20 0CB0	JSR-ABS	BC0C	INT, INT((100-Y)/8) in FPAC (R).

50047	C37F	A9 23	LDA-IMM	23	La9e byte adres 9etal 320.
50049	C381	A0 C9	LDY-IMM	C9	Ho9e byte adres 9etal 320.
50051	C383	20 28BA	JSR-ABS	BA28	MULTIPLY, 320*R in FPAC.
50054	C386	A9 28	LDA-IMM	28	La9e byte variabele adres A.
50056	C388	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres A.
50058	C38A	20 67B8	JSR-ABS	B867	ADD, A+320*R in FPAC.
50061	C38D	A2 28	LDX-IMM	28	La9e byte variabele adres A.
50063	C38F	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres A.
50065	C391	20 D4BE	JSR-ABS	BE04	320*R+L in variabele A.
50068	C394	A9 05	LDA-IMM	05	La9e byte variabele adres voor X.
50070	C396	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor X.
50072	C398	20 8CBA	JSR-ABS	BA8C	X in HLPREG.
50075	C39B	A9 14	LDA-IMM	14	La9e byte adres 9etal 8.
50077	C39D	A0 C9	LDY-IMM	C9	Ho9e byte adres 9etal 8.
50079	C39F	20 A2BB	JSR-ABS	BBA2	"8" in FPAC.
50082	C3A2	20 0CBB	JSR-ABS	BB0C	DIVIDE, X/8 in FPAC.
50085	C3A5	20 CCBC	JSR-ABS	BC0C	INT, INT(X/8) in FPAC (K).
50088	C3A8	A9 14	LDA-IMM	14	La9e byte adres 9etal 8.
50090	C3AA	A0 C9	LDY-IMM	C9	Ho9e byte adres 9etal 8.
50092	C3AC	20 28BA	JSR-ABS	BA28	MULTIPLY, 8*K in FPAC.
50095	C3AF	A9 28	LDA-IMM	28	La9e byte variabele adres voor A.
50097	C3B1	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor A.
50099	C3B3	20 67B8	JSR-ABS	B867	ADD, R*320+K*8+L in FPAC.
50102	C3B6	A9 1E	LDA-IMM	1E	La9e byte adres voor 9etal 8192.
50104	C3B8	A0 C9	LDY-IMM	C9	Ho9e byte adres voor 9etal 8192.
50106	C3BA	20 67B8	JSR-ABS	B867	ADD, 8192+R*320+K*8+L in FPAC (AD).
50109	C3BD	20 F7B7	JSR-ABS	B7F7	AD in \$ 0014 en \$ 0015.
50112	C3C0	A9 05	LDA-IMM	05	La9e byte variabele adres voor X.
50114	C3C2	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor X.
50116	C3C4	20 A2BB	JSR-ABS	BBA2	Variabele X in FPAC.
50119	C3C7	A9 19	LDA-IMM	19	La9e byte adres voor 9etal 7.
50121	C3C9	A0 C9	LDY-IMM	C9	Ho9e byte adres voor 9etal 7.
50123	C3CB	20 8CBA	JSR-ABS	BA8C	9etal 7 in HLPREG.
50126	C3CE	20 E9AF	JSR-ABS	AFE9	AND, X AND 7 in FPAC.
50129	C3D1	20 53B8	JSR-ABS	B853	SUBTRACT, 7-(X AND 7) in FPAC (BI).
50132	C3D4	20 9BBC	JSR-ABS	BC9B	INT, INT(BI) in FPAC voor POKE.
50135	C3D7	A6 65	LDX-Z.PAGE		BI, exPonent van 2, in X-register.
50137	C3D9	A9 00	LDA-IMM	00	Le9e ACCU.
50139	C3DB	38	SEC-IMPL		Carry bit "1".
50140	C3DC	2A	ROL-ACCU		2↑X in de ACCU.
50141	C3DD	0A	DEX-IMPL		X=X-1.
50142	C3DE	10 FC	BPL-REL	C3DC	Voor de volgende lus tot X=-1.
50144	C3E0	A0 00	LDY-IMM	00	
50146	C3E2	11 14	ORA-(IND),Y		Voor kruisende lijnen.
50148	C3E4	91 14	STA-(IND),Y		2↑BI in het bit map geheugen.
50150	C3E6	60	RTS-IMPL		RETURN.

Dit programmadeel is als een subroutine uitgevoerd en is ten opzichte van de andere programma's in dit boek nogal omvangrijk. Dat betekent nog niet dat het moeilijk is te volgen. Het is niet meer dan een aaneenschakeling van reeds behandelde routines voor het maken van berekeningen. Eerst volgt nu een lijstje van de indeling van de variabelen en de constanten van dit programma:

Y	C900
X	C905
100	C90F
8	C914

7	C919
8192	C91E
320	C923
A	C928
(100-Y)	C9AD

Om te beginnen wordt het adres AD berekend, aanvangend met de berekening van 100-Y. De waarde hiervan hebben we later nog een keer nodig, vandaar dat hij wordt bewaard in \$ C9AD. Direct hierna volgt het transport van 100-Y in FPAC naar HLPREG (regel 50007). Hiervoor is de subroutine op \$ BCOC gebruikt. Voor transport in

de tegenovergestelde richting is overigens ook een subroutine ter beschikking, van HLPREG naar FPAC op adres \$ BBFC. De AND bewerking van (100-Y) met 7 volgt hierna. Het resultaat L hiervan, dat een deel van AD is, wordt bewaard in het variabele-adres voor A. Het programma vervolgt met de berekening van R die na vermenigvuldiging met 320 bij L kan worden opgeteld. Na het berekenen van $K = \text{INT}(X/8)$ en het vermenigvuldigen van de uitkomst hiervan met 8 kan door optelling bij de waarde in A en bij 8192 het adres worden gevormd dat met de subroutine \$ B7F7 (INT) in de geheugenplaatsen \$ 0014 en \$ 0015 wordt geplaatst. Voor het berekenen van BI zijn heel wat minder programmaregels nodig.

Na de INT op BC9B voor het POKE getal in \$ 0065 kunnen we BI in het X-register laden (regel 50135). Om het getal te vinden dat in het adres AD moet worden geplaatst moet 2 BI worden gevormd. Dit doen we door met X keer ROL een 1 in bitnummer X te plaatsen ($X=0$, 1 in bit 0; $X=1$, 1 in bit 1; $X=2$, 1 in bit 2 enzovoorts). Hiervoor moet eerst het carry bit 1 worden (SEC, regel 50139).

Bij meerdere lijnen in één figuur kan van het getal in de geheugenplaats AD al een bit 1 zijn (als daar de lijnen elkaar kruisen). Om dat bit niet verloren te laten gaan wordt eerst de ORA bewerking met dit getal uitgevoerd en wordt BI daarna in AD geschreven.

Het verhogen van de X- of de Y-variabele

49939	C313	A5 FD	LDA-Z.PAGE		Lage byte variabele adres.
49941	C315	A0 C9	LDY-IMM	C9	Hoge byte variabele adres.
49943	C317	20 A2BB	JSR-ABS	BBA2	Variabele in FPAC.
49946	C31A	A9 0A	LDA-IMM	0A	Lage byte variabele adres voor D.
49948	C31C	A0 C9	LDY-IMM	C9	Hoge byte variabele adres voor D.
49950	C31E	20 67B8	JSR-ABS	B867	ADD, som in FPAC.
49953	C321	A6 FD	LDX-Z.PAGE		Lage byte variabele adres.
49955	C323	A0 C9	LDY-IMM	C9	Hoge byte variabele adres.
49957	C325	20 D4BB	JSR-ABS	BBD4	SOM uit FPAC in variabele adres.
49960	C328	60	RTS-IMPL		RETURN.

Bij elk programma voor het schrijven van een lijn op het scherm is het nodig dat of de X- of de Y-waarde wordt verhoogd. Hiertoe dient bovenstaande subroutine. Van de variabele die moet worden opgehoogd (X of Y) moet de lage byte (pointer) in geheugenplaats \$ OOFD worden gebracht. De grootte van de verhoging (hangt van de

functie af) moet in de floating point variabele D (\$ C90A) worden opgeslagen.

De bovenstaande programmadelen zijn allemaal nodig bij elk programma voor het tekenen van een grafiek. Voor het tekenen van een sinusfunctie moeten we aan dit 'basisprogramma' nog de volgende regels toevoegen:

*Routine voor $Y = 90 * \sin(X/50)$*

49554	C192	A2 39	LDX-IMM	39	CHR\$(57), voor "9".
49556	C194	8E 00C8	STX-ABS	C800	"9" in de buffer.
49559	C197	A2 30	LDX-IMM	30	CHR\$(48), voor "0".
49561	C199	8E 01C8	STX-ABS	C801	"0" in de buffer.
49564	C19C	A9 02	LDA-IMM	02	Voor 2 karakters.
49566	C19E	A2 32	LDX-IMM	32	Lage byte variabele adres.
49568	C1A0	86 FD	STX-Z.PAGE		Variabele Pointer.
49570	C1A2	20 00C3	JSR-ABS	C300	Getal 90 in variabele adres \$ C932.
49573	C1A5	A2 35	LDX-IMM	35	CHR\$(53), voor "5".
49575	C1A7	8E 00C8	STX-ABS	C800	"5" in de buffer.
49578	C1AA	A9 02	LDA-IMM	02	Voor 2 karakters.
49580	C1AC	A2 37	LDX-IMM	37	Lage byte variabele adres.
49582	C1AE	86 FD	STX-Z.PAGE		Variabele Pointer.
49584	C1B0	20 00C3	JSR-ABS	C300	Getal 50 in variabele adres \$ C937.
49587	C1B3	A2 31	LDX-IMM	31	CHR\$(49), voor "1".
49589	C1B5	8E 00C8	STX-ABS	C800	"1" in de buffer.
49592	C1B8	A9 01	LDA-IMM	01	Voor 1 karakter.
49594	C1BA	A2 0A	LDX-IMM	0A	Lage byte variabele adres voor D.
49596	C1BC	86 FD	STX-Z.PAGE		Variabele Pointer.

49598	C1BE	20	00C3	JSR-ABS	C300	Getal 1 in variabele adres voor D.
49601	C1C1	A2	30	LDX-IMM	30	CHR\$(48), voor "0".
49603	C1C3	8E	00C8	STX-ABS	C800	"0" in de buffer.
49606	C1C6	A9	01	LDA-IMM	01	Voor 1 karakter.
49608	C1C8	A2	05	LDX-IMM	05	Laag byte variabele adres voor X.
49610	C1CA	86	FD	STX-Z.PAGE		Variabele Pointer.
49612	C1CC	20	00C3	JSR-ABS	C300	Getal 0 in variabele adres voor X.
49615	C1CF	A9	05	LDA-IMM	05	Laag byte variabele adres voor X.
49617	C1D1	A0	C9	LDY-IMM	C9	Hog byte variabele adres voor X.
49619	C1D3	20	8CBA	JSR-ABS	BA8C	Variabele X in HLPREG.
49622	C1D6	A9	37	LDA-IMM	37	Laag byte adres getal 50.
49624	C1D8	A0	C9	LDY-IMM	C9	Hog byte adres getal 50.
49626	C1DA	20	A2BB	JSR-ABS	BBA2	Getal 50 in FPAC.
49629	C1DD	20	0CBB	JSR-ABS	BB0C	DIVIDE, X/50 in FPAC.
49632	C1E0	20	6BE2	JSR-ABS	E26B	SIN, SIN(X/50) in FPAC.
49635	C1E3	A9	32	LDA-IMM	32	Laag byte adres getal 90.
49637	C1E5	A0	C9	LDY-IMM	C9	Hog byte adres getal 90.
49639	C1E7	20	28BA	JSR-ABS	BA28	MULTIPLY, 90*SIN(X/50) in FPAC.
49642	C1EA	20	CCBC	JSR-ABS	BCCC	INT, INT(90*SIN(X/50)) in FPAC.
49645	C1ED	A2	00	LDX-IMM	00	Laag byte variabele adres voor Y.
49647	C1EF	A0	C9	LDY-IMM	C9	Hog byte variabele adres voor Y.
49649	C1F1	20	D4BB	JSR-ABS	BBD4	FPAC in variabele adres voor Y.
49652	C1F4	20	3FC3	JSR-ABS	C33F	Zet de stip op het scherm aan.
49655	C1F7	A2	05	LDX-IMM	05	Laag byte variabele adres voor X.
49657	C1F9	86	FD	STX-Z.PAGE		Variabele Pointer.
49659	C1FB	20	13C3	JSR-ABS	C313	Verhoog de variabele X met 1.
49662	C1FE	A9	05	LDA-IMM	05	Laag byte variabele adres voor X.
49664	C200	A0	C9	LDY-IMM	C9	Hog byte variabele adres voor X.
49666	C202	20	8CBA	JSR-ABS	BA8C	Variabele X in HLPREG.
49669	C205	A9	23	LDA-IMM	23	Laag byte adres getal 320.
49671	C207	A0	C9	LDY-IMM	C9	Hog byte adres getal 320.
49673	C209	20	A2BB	JSR-ABS	BBA2	Getal 320 in FPAC.
49676	C20C	20	53B8	JSR-ABS	B853	SUBTRACT, X-320 in FPAC.
49679	C20F	A5	66	LDA-Z.PAGE		Bepaal teken (+ of -) van het verschil.
49681	C211	10	AE	BPL-REL	C1C1	Herhaal de tekening van de grafiek.
49683	C213	20	3FC3	JSR-ABS	C33F	Zet de tweede stip op het scherm aan.
49686	C216	4C	CFC1	JMP-ABS	C1CF	Voor de volgende stippen.

Voor het tekenen van deze grafiek zijn twee punten van belang:

X moet veranderen van 0 tot 320 met stappen van 1 (BASIC: FOR X=0 TO 320 STEP 1) en voor een behoorlijke grafiek dient de formule

$$Y = 90 * \sin(X/50)$$

te worden verwezenlijkt. Het getal 90 geeft de maximale uitwijking in Y richting. Deze kan op het scherm niet groter zijn dan 100. Bij X=314 is X/50 gelijk aan 2π zodat een volledige sinus figuur op het scherm kan worden geschreven. Om te beginnen zullen de getallen D=1 (step 1) 90 en 50 moeten worden ingevoerd. Hiervoor zijn de volgende geheugenplaatsen gereserveerd:

D	C90A
90	C932
50	C937

Het invoeren hiervan geschiedt in de regels 49554 tot en met 49598 op de gebruikelijke wijze. Omdat eerst 90 is ingevoerd hoeft daarna alleen maar het karakter voor 5 in de buffer te worden geplaatst. Het karakter voor 0 is daar al.

Daarnaast zullen de grenzen moeten worden ingevoerd waartussen de X-waarde moet veranderen voor het tekenen van de lijn (FOR X=0 TO 320). Het getal 0 wordt ingevoerd in de variabele voor X (regels 49601 tot en met 49612). De waarde 320 is al ingevoerd (\$ C923).

Nu vangt het berekenen van $Y=90*\sin(X/50)$ aan, geheel volgens de bekende methode. Als de waarde voor Y is gevonden wordt deze in de geheugenplaats voor Y geladen (regels 49645 tot en met 49649).

Nu kan in de subroutine op adres \$ C33F het punt op het scherm, behorende bij die X- en Y-waarde, worden 'aangezet'. Door dit steeds voor een hoger waarde voor X te herhalen wordt de lijn op het

scherm getekend. Nu zal een dergelijke lijn op een normaal televisietoestel slecht of bijna niet te zien zijn. Beter is het om twee punten naast elkaar aan te zetten. Daarom wordt na het aanzetten van het eerste punt ook nog een tweede punt berekend, met een hoger X-waarde maar met dezelfde Y-waarde. Daarvoor wordt eerst X opgehoogd in de subroutine op § C313.

Deze moet dan wel weten welke variabele moet worden opgehoogd (regels 49655 en 49657). Het punt $X=320$ zou net buiten het scherm vallen.

Het hoogste regelnummer is 319. Daarom wordt X eerst vergeleken met 320 ($X-320$). Met LDA Z.PAGE \$ 0066 wordt het teken van het resultaat bepaald. Zolang dit negatief is wordt het programma voortgezet met het printen van een punt met de nieuwe X-waarde maar de oude Y-waarde (regel 49683) om een heldere lijn te krijgen. Dan wordt teruggegaan voor het bepalen van een punt met een nieuwe Y-waarde. Is X gelijk aan 320 geworden dan wordt geen nieuw punt meer aangezet. Het programma herhaalt steeds het tekenen van dezelfde lijn.

Het laatste voorbeeld van een machinetaalprogramma is het tekenen van een rechte lijn op het beeldscherm.

Het meest eenvoudig zijn de horizontale en de verticale lijnen te tekenen. Voor de horizontale lijn geldt de uitdrukking $Y=k$, waarbij k een bepaald getal is ($k=30$, $Y=30$; $k=-70$, $Y=-70$). De lijn wordt getekend door voor Y het constante getal in de variabele plaats in te voeren en X vanaf zijn minimum waarde te verhogen tot zijn maximum waarde. Voor de verticale lijn geldt $X=k$. In dat geval wordt voor X een constant getal ingevoerd

en laten we Y toenemen van zijn minimum waarde tot zijn maximum waarde. Voor lijnen die onder een bepaalde hoek staan ten opzichte van de X-as geldt de algemene formule:

$$Y = a \cdot X + b$$

Hierin zijn a en b constanten. Bijvoorbeeld: $Y = -3 \cdot X + 90$. Hierin is a gelijk aan -3 en b gelijk aan 90. Een lijn waarvan de absolute waarde van a gelijk is aan 1 ($Y = X + b$ of $Y = -X + b$) vormt een scherpe hoek met de X-as van 45° . Voor het tekenen van een lijn waarvan de scherpe hoek die gevormd wordt met de X-as kleiner is dan 45° ($|a| < 1$) kan de formule $Y = a \cdot X + b$ worden gebruikt voor het berekenen van Y, waarbij X steeds met 1 moet worden opgehoogd van zijn minimum waarde tot de maximum waarde. Passen we dit ook toe op lijnen waarvan de scherpe hoek tussen de lijn en de X-as groter is dan 45° ($|a| > 1$) dan wordt deze vaak brokkelig weergegeven, vooral als deze hoek de 90° benadert. Voor deze lijnen kan $Y = a \cdot X + b$ worden veranderd in $X = b/a - Y/a$.

We laten nu steeds X berekenen voor waarden van Y die van minimum naar maximum steeds 1 groter zijn dan de voorafgaande. Het volgende programma demonstreert dit met de lijnen

$$Y = X/3 - 90 \quad \text{en} \quad Y = -3 \cdot X + 90$$

De eerste lijn vormt een hoek met de X-as kleiner dan 45° ($|a| = 1/3$) en de tweede lijn groter dan 45° ($|a| = 3$). Eerst de lijn $Y = X/3 - 90$, waarvoor tevens geldt:

FOR X=10 TO 300 STEP 1.

Routine voor $Y = X/3 - 90$

49554	C192	A9 33	LDA-IMM	33	CHR\$(51), voor "3".
49556	C194	8D 00C8	STA-ABS	C800	"3" in de buffer.
49559	C197	A9 30	LDA-IMM	30	CHR\$(48), voor "0".
49561	C199	8D 01C8	STA-ABS	C801	"0" in de buffer.
49564	C19C	8D 02C8	STA-ABS	C802	"0" in de buffer.
49567	C19F	A9 03	LDA-IMM	03	Voor 3 karakters.
49569	C1A1	A2 2D	LDX-IMM	2D	Laag byte variabele adres.
49571	C1A3	86 FD	STX-Z.PAGE		Variabele Pointer.
49573	C1A5	20 00C3	JSR-ABS	C300	Getal 300 in variabele adres \$ C92D.
49576	C1A8	A9 02	LDA-IMM	02	Voor 2 karakters.
49578	C1AA	A2 3C	LDX-IMM	3C	Laag byte variabele adres.
49580	C1AC	86 FD	STX-Z.PAGE		Variabele Pointer.
49582	C1AE	20 00C3	JSR-ABS	C300	Getal 30 in variabele adres \$ C93C.
49585	C1B1	A9 01	LDA-IMM	01	Voor 1 karakter.
49587	C1B3	A2 32	LDX-IMM	32	Laag byte variabele adres.
49589	C1B5	86 FD	STX-Z.PAGE		Variabele Pointer.

49591	C1B7	20	00C3	JSR-ABS	C300	Getal 3 in variabele adres \$ C932.
49594	C1BA	A9	39	LDA-IMM	39	CHR\$(57), voor "9".
49596	C1BC	8D	00C8	STA-ABS	C800	"9" in de buffer.
49599	C1BF	A9	02	LDA-IMM	02	Voor 2 karakters.
49601	C1C1	A2	37	LDX-IMM	37	La9e byte variabele adres.
49603	C1C3	86	FD	STX-Z.PAGE		Variabele Pointer.
49605	C1C5	20	00C3	JSR-ABS	C300	Getal 90 in variabele adres \$ C937.
49608	C1C8	A9	31	LDA-IMM	31	CHR\$(49), voor "1".
49610	C1CA	8D	00C8	STA-ABS	C800	"1" in de buffer.
49613	C1CD	A9	02	LDA-IMM	02	Voor 2 karakters.
49615	C1CF	A2	05	LDX-IMM	05	La9e byte variabele adres voor X.
49617	C1D1	86	FD	STX-Z.PAGE		Variabele Pointer.
49619	C1D3	20	00C3	JSR-ABS	C300	Getal 10 in variabele X.
49622	C1D6	A9	01	LDA-IMM	01	Voor 1 karakter.
49624	C1D8	A2	0A	LDX-IMM	0A	La9e byte variabele adres voor D.
49626	C1DA	86	FD	STX-Z.PAGE		Variabele Pointer.
49628	C1DC	20	00C3	JSR-ABS	C300	Getal 1 in variabele D.
49631	C1DF	A9	32	LDA-IMM	32	La9e byte adres voor getal 3.
49633	C1E1	A0	C9	LDY-IMM	C9	Ho9e byte adres voor getal 3.
49635	C1E3	20	A2BB	JSR-ABS	BBA2	Getal 3 in FPAC.
49638	C1E6	A9	05	LDA-IMM	05	La9e byte variabele adres voor X.
49640	C1E8	A0	C9	LDY-IMM	C9	Ho9e byte variabele adres voor X.
49642	C1EA	20	8CBA	JSR-ABS	BA8C	Variabele X in HLPREG.
49645	C1ED	20	0CBB	JSR-ABS	BB0C	DIVIDE, X/3 in FPAC.
49648	C1F0	20	0CBC	JSR-ABS	BC0C	FPAC naar HLPREG, X/3 in HLPREG.
49651	C1F3	A9	37	LDA-IMM	37	La9e byte adres getal 90.
49653	C1F5	A0	C9	LDY-IMM	C9	Ho9e byte adres getal 90.
49655	C1F7	20	A2BB	JSR-ABS	BBA2	Getal 90 in FPAC.
49658	C1FA	20	53BB	JSR-ABS	B853	SUBTRACT, X/3-90 in FPAC.
49661	C1FD	A2	00	LDX-IMM	00	La9e byte variabele adres voor Y.
49663	C1FF	A0	C9	LDY-IMM	C9	Ho9e byte variabele adres voor Y.
49665	C201	20	D4BB	JSR-ABS	BBD4	X/3-90 in variabele Y.
49668	C204	20	3FC3	JSR-ABS	C33F	Zet de stip op het scherm aan.
49671	C207	A9	05	LDA-IMM	05	La9e byte variabele adres voor X.
49673	C209	85	FD	STA-Z.PAGE		Variabele Pointer.
49675	C20B	20	13C3	JSR-ABS	C313	Verhoog X met 1.
49678	C20E	20	3FC3	JSR-ABS	C33F	Zet de tweede stip aan.
49681	C211	A9	2D	LDA-IMM	2D	La9e byte adres getal 300.
49683	C213	A0	C9	LDY-IMM	C9	Ho9e byte adres getal 300.
49685	C215	20	A2BB	JSR-ABS	BBA2	Getal 300 in FPAC.
49688	C218	A9	05	LDA-IMM	05	La9e byte adres voor de variabele X.
49690	C21A	A0	C9	LDY-IMM	C9	Ho9e byte adres voor de variabele X.
49692	C21C	20	8CBA	JSR-ABS	BA8C	Variabele X in HLPREG.
49695	C21F	20	53BB	JSR-ABS	B853	SUBTRACT, X-300 in FPAC.
49698	C222	A5	66	LDA-Z.PAGE		BePaal teken (+/-) van het verschil.
49700	C224	30	B9	BMI-REL	C1DF	Terug voor volgend punt.

Dit programma moet samen met het basisprogramma worden gebruikt.

De eerste regel begint dan ook op adres \$ C192 en sluit daarmee aan op het basisprogramma. Eerst worden er weer een aantal constanten ingevoerd (de getallen 300, 3, 90, 10 en 1; regels 49554 tot en met 49628). Dit gebeurt op de gebruikelijke wijze. Daarna volgt de berekening voor Y, geheel op de manier zoals hiervoor al eens is gedemonstreerd (regels 49631 tot en met 49665). Opgemerkt wordt dat het toepassen van de subroutine DIVIDE op deze manier in programma's wel eens een negatief resultaat kan opleveren. Is dat zo dan kan dat op-

gelost worden door in een andere volgorde de registers FPAC en HLPREG in te vullen met de juiste getallen (hier 3 in FPAC en X in HLPREG). Uiteraard is de volgorde voor dit programma juist.

Eerst FPAC laden en daarna HLPREG. Na het berekenen van de waarde van Y (afhankelijk van X) wordt de stip op het scherm aangezet. Daarna wordt X opgehoogd met D=1 en wordt met dezelfde Y-waarde opnieuw een stip op het scherm aangezet (twee stippen naast elkaar). Nu wordt X met het getal 300 vergeleken. Zolang X kleiner is wordt het tekenen voortgezet met steeds een nieu-

we waarde voor X. Als X gelijk is aan 300 komen we terecht in het programmagedeelte voor het te-

kenen van de lijn $Y = -3 \cdot X + 90$ waarvoor geldt: $X = 30 - Y/3$; FOR $Y = -90$ TO 90 STEP 1.

Routine voor $Y = -3 \cdot X + 90$

49702	C226	A9 37	LDA-IMM	37	La9e byte adres getal 90.
49704	C228	A0 C9	LDY-IMM	C9	Ho9e byte adres getal 90.
49706	C22A	20 A2BB	JSR-ABS	BBA2	Getal 90 in FPAC.
49709	C22D	20 B4BF	JSR-ABS	BFB4	NEG, getal -90 in FPAC.
49712	C230	A2 00	LDX-IMM	00	La9e byte variabele adres voor Y.
49714	C232	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor Y.
49716	C234	20 D4BB	JSR-ABS	BBD4	Getal -90 in variabele adres voor Y.
49719	C237	A9 32	LDA-IMM	32	La9e byte adres voor getal 3.
49721	C239	A0 C9	LDY-IMM	C9	Ho9e byte adres voor getal 3.
49723	C23B	20 A2BB	JSR-ABS	BBA2	Getal 3 in FPAC.
49726	C23E	A9 00	LDA-IMM	00	La9e byte variabele adres voor Y.
49728	C240	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor y.
49730	C242	20 8CBA	JSR-ABS	BA8C	Variabele Y in HLPREG.
49733	C245	20 8CBB	JSR-ABS	BB0C	DIVIDE, Y/3 in FPAC.
49736	C248	A9 3C	LDA-IMM	3C	La9e byte adres getal 30.
49738	C24A	A0 C9	LDY-IMM	C9	Ho9e byte adres getal 30.
49740	C24C	20 8CBA	JSR-ABS	BA8C	Getal 30 in HLPREG.
49743	C24F	20 53B8	JSR-ABS	B853	SUBTRACT, $30 - Y/3$ in FPAC.
49746	C252	A2 05	LDX-IMM	05	La9e byte variabele adres voor X.
49748	C254	A0 C9	LDY-IMM	C9	Ho9e byte variabele adres voor X.
49750	C256	20 D4BB	JSR-ABS	BBD4	$30 - Y/3$ in adres voor variabele X.
49753	C259	20 3FC3	JSR-ABS	C33F	Zet de stip op het scherm aan.
49756	C25C	A9 05	LDA-IMM	05	La9e byte variabele adres voor X.
49758	C25E	85 FD	STA-Z.PAGE		Variabele Pointer.
49760	C260	20 13C3	JSR-ABS	C313	Verhoog X met 1.
49763	C263	20 3FC3	JSR-ABS	C33F	Zet de tweede stip op het scherm aan.
49766	C266	A9 00	LDA-IMM	00	La9e byte variabele adres voor Y.
49768	C268	85 FD	STA-Z.PAGE		Variabele Pointer.
49770	C26A	20 13C3	JSR-ABS	C313	Verhoog Y met 1, Y in FPAC.
49773	C26D	A9 37	LDA-IMM	37	La9e byte adres voor getal 90
49775	C26F	A0 C9	LDY-IMM	C9	Ho9e byte adres voor getal 90
49777	C271	20 8CBA	JSR-ABS	BA8C	Getal 90 in HLPREG.
49780	C274	20 53B8	JSR-ABS	B853	SUBTRACT, $90 - Y$ in FPAC.
49783	C277	A5 66	LDA-Z.PAGE		Bepaal het teken van het verschil.
49785	C279	10 BC	BPL-REL	C237	Voor een volgend Punt.
49787	C27B	4C C8C1	JMP-ABS	C1C8	Teken de lijnen opnieuw.

Eerst moet Y met het getal -90 worden geladen (FOR $Y = -90$ TO 90). Dit wordt gedaan door de negatieve waarde van het getal 90 te nemen en dit in Y te plaatsen (regels 49702 tot en met 49716). De berekening van de waarde voor X zien we in de regels 49719 tot en met 49750. Het getal 30 was al in het eerste gedeelte van het programma ingevoerd. Nu wordt de stip op het scherm aangezet en

wordt X opgehoogd voor de tweede stip (dit is bij deze lijn *meer* nodig dan bij de vorige). Nu moet Y worden opgehoogd voor het berekenen van een nieuwe X-waarde en het printen van twee nieuwe stippen. Als laatste volgt het vergelijken van Y met het getal 90. Is deze waarde bereikt dan wordt teruggegaan naar het begin voor het opnieuw tekenen van de lijnen.

GEBRUIKTE SUBROUTINES

Subroutine	Adres	Pagina	Subroutine	Adres	Pagina
ABS	BC58	79	LOAD	FFD5	68
ADD	B867	77	LOG	B9EA	79
AND	AFE9	78	MULTIPLY	BA28	77
ASCII-FLOTING POINT	B7B5	63	NEG	BFB4	79
ATN	E3OE	79	OPEN	FFCO	69
CHKIN	EFC6	70	OR	AFE6	78
CHKOUT	FFC9	69	PEEK(X) IN FPAC	B80D	65
CHRIN	FFCF	63	PLOT	FFF0	84
CHROUT	FFD2	57	POWER	BF7B	78
CLALL	FFE7	69	PRINT FILE	F5C1	62
CLEAR SCREEN	E544	85	PRINT		
CLOSE	FFC3	69	KARAKTERSTRING	AB1E	62
COS	E264	79	PRINT		
DELAY 1 ms	EEB3	86	"ILL. QUANT. ERR."	B248	65
DIVIDE	BBOC	78	RND	E097	79
EXP	BFED	79	SAVE	FFD8	68
FLOTING POINT-ASCII	BDDD	64	SCNKEY	FF9F	59
FPAC naar HLPREG	BCOC	99	SETLFS	FFBA	67
FPAC-VARIAB. ADRES	BBD4	63	SETNAM	FFBD	68
GETIN	FFE4	56	SGN	BC39	79
HLPREG naar FPAC	BBFC	99	SIN	E26B	79
HOME	E566	85	SQR	BF71	79
INT (FPAC) in FPAC	BCCC	79	SUBTRACT	B853	78
INT (FPAC) voor POKE	BC9B	65	TAN	E2B4	79
INT (FPAC)-			VARIAB. ADRES-FPAC	BBA2	64
POKE ADR.	B7F7	65	VARIAB. IN HLPREG	BA8C	78

OVERZICHT LIJSTEN

Lijstnr.	Inhoud	Pagina	Lijstnr.	Inhoud	Pagina
1	Hexadecimale getallen	10	6	SCAN-getallen van het toetsenbord	60
2	Instructieset 6510 processor	45	7	Rekenkundige bewerkingen	77
3	Aanvangsadres geheugensectie	52	8	Rekenkundige en logische bewerkingen	78
4	Aanvangsadres schermgeheugen	53	9	Functionies	79
5	Aanvangsadres karaktergenerator	54			

GEBRUIKTE PROGRAMMA'S

Programma	Pagina
Het in- en uitschakelen van de BASIC interpreter.....	50
Een subroutine achter de BASIC interpreter	51
Gebruik van de subroutine achter de BASIC interpreter.....	51
Keuze van de geheugensectie.....	52
Selectie van sectie, schermgeheugen en karaktergenerator	54
GET karakter; PRINT karakter	57
Het verlaten van een programma	57
GET karakterstring	57
Wacht op een toets in.....	59
Wacht op een toets los	59
Wacht op een bepaalde toets in	60
GET A\$	61
PRINT FILE.....	62
INPUT A	62
PRINT A	63
INPUT routine	64
POKE X, Y	64
PRINT PEEK (X)	65
SAVE een programma	67
LOAD een programma.....	68
SAVE een file naar de disk	69
LOAD een file van de disk	70
SAVE een file naar de tape	71
LOAD een file van de tape.....	71

Programma	Pagina
File naar de printer	72
HARD COPY	73
Rekenkundige bewerkingen I	76
Rekenkundige bewerkingen II.....	77
Functies I.....	78
Functies II	78
PRINT 475*SIN ($\pi/6$)	80
PRINT INT (6*RND(1)) + 1.....	81
USER functie	82
Bepaal de plaats van de cursor.....	84
Besturing van de cursor	84
Bewegende beelden	85
Kaatsende bal	86
Bewegende sprite.....	88
Scrolling, INIT.....	91
Vul het beeldscherm.....	92
Scrolling 1 kolom.....	92
Vul kolom 39.....	93
Scrolling, subroutines.....	93
Hoge resolutie, INIT	96
Hoge resolutie, invoeren van constanten	97
Berekening van het adres en BI	98
Het verhogen van de X- of de Y-variabele	100
Routine voor $Y = 90 * \sin(X/50)$	100
Routine voor $Y = X/3 - 90$	102
Routine voor $Y = -3 * X + 90$	104

Commodore 64, programmeren in machinetaal.

Voor de beginners:

Weet u niets of slechts weinig van programmeren in machinetaal, dan vindt u in dit boek alles om volledig te worden ingewijd in deze techniek en de mogelijkheden die de Commodore 64 u biedt.

Voor de gevorderde machinetaalprogrammeur:

Dit boek geeft u een duidelijke verklaring van de werkingen binnen de 6510 processor. Bij de diverse instructies en adresseermethoden vindt u niet alleen een beschrijving van de voorbeelden, ook het gebruik van de subroutines uit het systeemprogramma en de BASIC interpreter komen aan de orde. Kortom alle informatie die nodig is om de mogelijkheden van uw Commodore 64 computer volledig te kunnen uitbuiten, komen in deze uitgave aan de orde.